ИВАН МИРЧЕВ

ГРАФИ

ОПТИМИЗАЦИОННИ АЛГОРИТМИ В МРЕЖИ



БЛАГОЕВГРАД, 2001 г.

В книгата на доц. д-р Ив. Мирчев се разглеждат въпроси от теория на графите и дискретното оптимиране. Представени са алгоритми, свързани с намирането на структурни и числови характеристики на графови обекти.

В първа глава са разгледани основни понятия от теория на графите - подграфи, матрично представяне на графи, дървета, разрези, цикли, хамилтонови графи, планарни и двойнствени графи, ребрени и върхови оцветявания и др. Дадени са някои класически резултати от теория на графите. Подходът е алгоритмичен и адресиран към читателя, интересуващ се от приложните аспекти на теория на графите.

Във втора глава са представени задачата на линейното оптимиране и алгоритми в мрежи, свързани с намирането на покриващи дървета, най-кратки пътища, потоци с минимална цена, критични пътища в мрежовото планиране и управление, разполагане на обекти, оптимални маршрути (задачи за китайския пощальон, за търговския пътник, за назначенията) и др. Разгледани са основни методи за анализ и търсене в графи, и въпросът за сложността на предложените алгоритми.

Алгоритмите в книгата са описани без изклишен формализъм и са илюстрирани с много примери, което я прави достъпна за широк кръг читатели - студенти, икономисти, инженери, стопански ръководители, физици, химици, математици, информатици и др.

Издателски съвет

ИВАН МИРЧЕВ

Графи. Оптимизационни алгоритми в мрежи

Първо издание

ISBN 954-680-174-7

Благоевград

Предговор

В много случаи описанието, анализът и изучаването на реални системи се осъществява успешно и сравнително просто с езика на графите. По-точно това изучаване е свързано с намирането на структурни и числови характеристики на обекти от теория на графите. Нагледността на теоретико-графовите структури позволява на естествен и достъпен език ефективно да се формализират и решават достатъчно сложни приложни задачи.

С езика на теория на графите успешно се описват проблеми в икономиката, автоматизираното управление на производството, планирането, гражданското и военното строителство, проектирането на съвременни изчислителни системи, електротехниката, квантовата теория на полето, генетичните системи и други области.

Предложената книга е адресирана към читателя, интересуващ се от приложния характер, изчислителните и алгоритмични аспекти на теория на графите.

В първа глава са дадени някои класически резултати и алгоритми от тази теория, като основно е разгледан въпросът за съществуването на тези алгоритми, без да се обсъжда и коментира тяхната сложност.

Във втора глава са дадени важни и перспективни алгоритмически методи в теория на графите и мрежите, като се отчита тяхната ефективност. Целта е да се формира у читателя цялостна и логически стройна представа за алгоритмите, свързани с анализа на графи.

Съдържанието в края на книгата дава подробно описание на проблемите, разгледани в нея.

Чисто приложният характер на тази книга обяснява липсата на излишна математическа строгост и формализъм на някои места, което я прави интересна и достъпна за широк кръг читатели — студенти, икономисти, инженери, математици, информатици и др.

Изложеният в книгата материал авторът използва като основа на лекционните курсове "Теория на графите" и "Оптимизационни алгоритми в графи и мрежи", предназначени за студенти от специалностите математика, физика и математика, информатика и стопанско управление в Югозападен Университет "Неофит Рилски" — Благоевград.

При изложението на материала авторът се е опитал да намери подход, съчетаващ в себе си несъмнените достойнства на [1], [2] и [3] от цитираната литература.

Искам да благодаря на сина ми Иван, баща ми и всички, които търпеливо ми вярваха и подкрепяха. Признателен съм на рецензентите Петър Миланов и Борислав Юруков за направените съдържателни препоръки, и на студентката Елена Драганова за техническата помощ при написването на тази книга.

Иван Мирчев

08.10.2001 г.
Благоевград

Глава 1

Теория на графите. Алгоритмичен подход

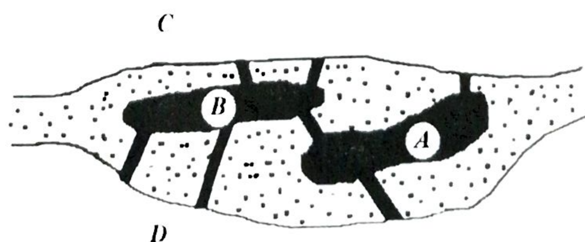
1.1. Основни понятия в теория на графите

Много транспортни, разпределителни задачи, задачи за избор на оптимални маршрути или разположение на центрове за обслужване, задачи, свързани с изготвяне на оптимални графици и разписания, се описват с езика на графите и мрежите. Най-общо казано, редица физически, химически, икономически и управленски системи успешно се интерпретират и изследват с теория на графите.

В много случаи тези задачи са линейни — линейна е както целевата функция, така и всяко от ограниченията, което означава, че те са решими с методите на линейното оптимиране. Реалните задачи (тези, които достатъчно пълно отразяват действителността) обаче са с твърде голяма размерност, поради което се налага да се търсят ефективни и гъвкави алгоритми по отношение изменението на изходните данни. Теорията на графите дава добри възможности за това.

През 1736 г. Леонард Ойлер публикува първата работа по теория на графите в Санкт Петербург, свързана с популярната задача за Кьонигсбергските мостове.

Кьонигсберг (Калининград), намиращ се тогава в Източна Прусия, бил построен на мястото на сливането на две реки и на два острова. Островите били съединени помежду си и с бреговете на реката чрез седем моста, както е показано на черт. 1.1.



Черт. 1.1

Въпросът бил, може ли жител на града, тръгвайки от дома си (кой да е от четирите участъка от сушата A, B, C, D) да премине по всеки от седемте моста точно по веднъж и да се върне у дома си.

Такъв род задачи - задачата за китайския пощальон, кой-то трябвало да премине по всички адреси, разнасяйки писмата, така че изминатият път да е минимален и да се върне там, от-където е тръгнал; Ойлеровата задача за мостовете; задачата на Уилям Хамилтон за обхождането на всички ребра на доде-каедър, минавайки през всеки връх само веднъж, поставят на-чалото на развитието на теория на графите. През миналия век развитието и използването на тази теория било свързано най-вече с биохимията и молекулярния строеж на веществата, както и с теорията на електрическите вериги и схеми. Последните 50 години на XX век решаването на оптимизационни задачи в гра-фи и мрежи получава широко развитие, обусловено от масовото използване на мощни компютърни системи.

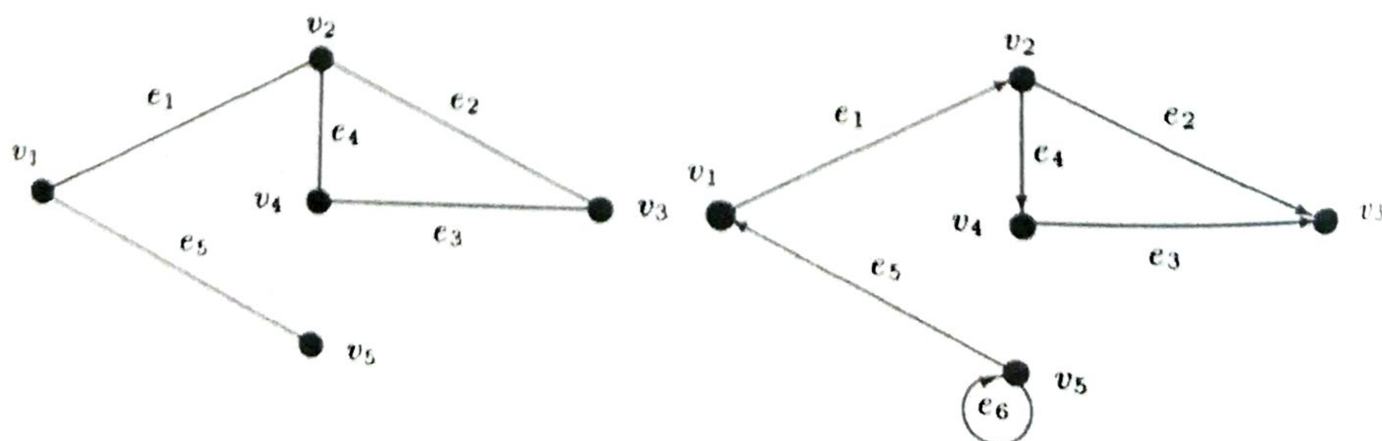
Какво е това *граф*? Графът е съвкупност от две групи елемен-ти - точки (*върхове*) и линии (*ребра*), съединяващи някои двойки точки. Най-често точките се изобразяват в равнина, а линии-те са прави, без това да е задължително. Ние ще разглеждаме графи, в които множеството V от върхове и множеството A от ребра са крайни. С други думи, ако

$V = \{v_1, v_2, ..., v_n\}$ е множество от върхове и

$A = \{(v_i, v_j) | v_i \in V,\ v_j \in V\}$ е множество от ребра,

съвкупността $G = (V, A)$ ще наричаме *неориентиран граф*. При това ако елементите на множеството A са *наредени двойки* (v_i, v_j), графът ще наричаме *ориентиран*, а елементите на множеството A ще наричаме *дъги*. За определеност по-нататък с $G = (V, A)$ и $G = (V, E)$ ще бележим съответно неориентиран и ориентиран граф.

ПРИМЕР 1.1.

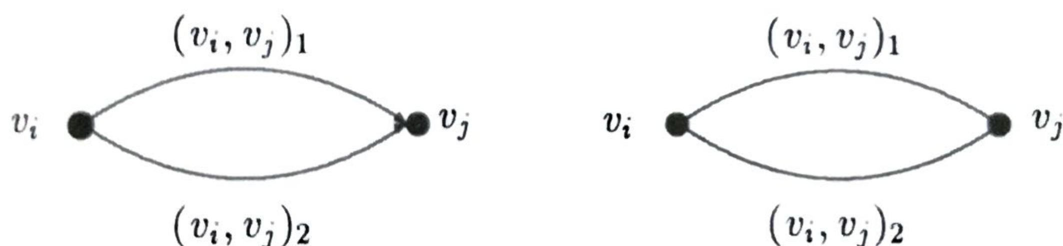


В пример 1.1 са изобразени неориентиран (вляво) и ориентиран граф (вдясно). Върховете v_1 и v_2 се наричат краища на реброто, респективно на дъгата (v_1, v_2). Върховете v_1 и v_2 се наричат съответно *начален и краен връх на дъгата*. Когато началният и крайният връх на дъгата съвпадат, дъгата се нарича още *примка*. Такава е дъгата e_6 на ориентирания граф в горния пример.

Ако в един ориентиран граф игнорираме посоката на дъгите, ще получим съответен неориентиран граф — *неориентиран дубликат, двойник*.

Един връх и една дъга (ребро) ще наричаме *инцидентни*, ако върхът се явява начален или краен за дъгата (край на реброто).

Когато два върха v_i, v_j са съединени с повече от една дъга (ребро) (v_i, v_j), дъгите (ребрата) се наричат *паралелни* и се обозначават с помощта на индекси. Например



Графите, в които съществуват паралелни дъги (v_i, v_j), обикновено се наричат *мултиграфи* и се явяват някакво обобщение на понятието граф. Ако най-големият брой паралелни дъги е s, то графът се нарича *s-граф*. Очевидно графът без паралелни дъги е 1-граф. S-графите описват реални физически, управленски и други системи. Например в електротехниката, изобразяването на електрическите вериги почти винаги се явява s-граф заради възможността паралелно да бъдат включени няколко електрически компоненти. В системите с гарантирана надеждност връзките между най-важните възли (устройства на системата) често са най-малко дублирани. Такива системи се описват с s-графи.

Две дъги (ребра) се наричат *съседни дъги (ребра)*, ако са инцидентни с един и същ връх на графа. Такива са например дъгите (ребрата) e_3 и e_4 от пример 1.1.

Два върха се наричат *съседни върхове*, ако съществува дъга (ребро), която ги съединява. Такива са върховете v_1 и v_2 от пример 1.1. Върховете v_4 и v_5 не са съседни.

Ориентираният граф може да се опише като се зададе множество V (от върхове) и релация Γ, дефинирана във V, която показва как са свързани върховете. В този случай графът се

обозначава като двойката $G = (V, \Gamma)$.

За ориентирания граф от пример 1.1, $V = \{v_1, v_2, v_3, v_4, v_5\}$ и $\Gamma(v_1) = \{v_2\}$; $\Gamma(v_2) = \{v_3, v_4\}$; $\Gamma(v_3) = \emptyset$; $\Gamma(v_4) = \{v_3\}$; $\Gamma(v_5) = \{v_1, v_5\}$, където $\Gamma(v_i)$ са върховете, явяващи се крайни за дъгите с начало v_i.

Когато графът е неориентиран или *смесен* (има и дъги и ребра), релацията Γ задава еквивалентен ориентиран граф, получаващ се от изходния чрез замяна на всяко ребро с двойка противоположни дъги.

Обратната релация $\Gamma^{-1}(v_i)$ задава множеството върхове v_s, за които в G съществува дъга (v_s, v_i). За графа от пример 1.1 например

$$\Gamma^{-1}(v_1) = \{v_5\}; \quad \Gamma^{-1}(v_2) = \emptyset; \quad \Gamma^{-1}(v_3) = \{v_2, v_4\} \quad \text{и т. н.}$$

Очевидно ако графът е неориентиран, за всяко $v_i \in V$

$$\Gamma^{-1}(v_i) = \Gamma(v_i).$$

Когато $V_t = \{v_1, v_2, \ldots, v_t\}$ е множество от върхове, под $\Gamma(V_t)$, се разбира

$$\Gamma(v_1) \cup \Gamma(v_2) \cup \ldots \cup \Gamma(v_t).$$

Освен това с $\Gamma^2(v_i)$ се бележи $\Gamma(\Gamma(v_i))$, с $\Gamma^3(v_i)$ се бележи $\Gamma(\Gamma(\Gamma(v_i)))$ и т.н. Аналогично се въвеждат означенията $\Gamma^{-2}(v_i)$; $\Gamma^{-3}(v_i)$ и т.н.

Броят на инцидентните с върха v_i дъги (ребра) се нарича *степен на върха* v_i и се бележи с $d(v_i)$. Дъгата (v_i, v_j) се нарича *изходяща* за върха v_i и *входяща* за върха v_j (излизаща от върха v_i и влизаща във върха v_j). С $d^-(v_i)$ ще бележим броя на входящите дъги за v_i, а с $d^+(v_i)$ — броя на изходящите за v_i дъги. Тези числа се наричат съответно *полустепен на входа и полустепен на изхода на върха*.

Върховете със степен 1 се наричат *висящи върхове* на графа, а върховете със степен 0 се наричат *изолирани върхове*.

От дадените определения очевидно следва верността на следните твърдения.

▷ *ТЕОРЕМА 1.1. Ако $G = (V, A)$ е произволен неориентиран граф, то:*

а) $\sum_{v_i \in V} d(v_i) = 2m$, *където m е броят на ребрата;*

б) броят на върховете с нечетна степен е четно число. ◁

Верността на а) очевидно следва от факта, че всяко ребро е инцидентно с точно два върха. Подточка б) следва от а), като вземем предвид релацията

$$\sum_{i=1}^{n} d(v_i) = \sum_{i=1}^{k} d(v_i) + \sum_{i=k+1}^{n} d(v_i), \tag{1.1}$$

където с $v_1, v_2, \ldots, v_k$ сме обозначили върховете с четна степен. Очевидно, щом сборът и едното събираемо в (1.1) са четни, то и другото събираемо $\sum_{i=k+1}^{n} d(v_i)$ е четно. Тъй като всички членове на тази сума са нечетни числа, то броят на събираемите в тази сума трябва да е четно число, което и трябваше да обосновем.

В случай на ориентиран граф $G = (V, E)$, очевидно имаме

1. $d(v_i) = d^+(v_i) + d^-(v_i)$;

2. $\sum_{v_i \in V} d^+(v_i) = \sum_{v_i \in V} d^-(v_i) = m$, където m е броят на дъгите.

От принципа на Дирихле "ако $n + 1$ предмета трябва да се поставят в n чекмеджета, то в поне едно чекмедже ще има поне два предмета" следва верността и на следното твърдение.

▷ *ТЕОРЕМА 1.2. Във всеки прост граф G (без паралелни дъги (ребра) и примки) съществуват поне два различни върха с една и съща степен, т.е. съществуват v_i и v_j, за които $d(v_i) = d(v_j)$.*

Доказателство: Ако всички върхове на такъв граф са изолирани или поне два върха са изолирани, твърдението е вярно. Нека неизолираните върхове на графа G са $n+1$ на брой — $v_1, v_2, \ldots, v_n, v_{n+1}$. Степента на всеки неизолиран връх е число от множеството $\{1, 2, \ldots, n\}$. От принципа на Дирихле следва, че поне два неизолирани върха ще имат една и съща степен. ◁

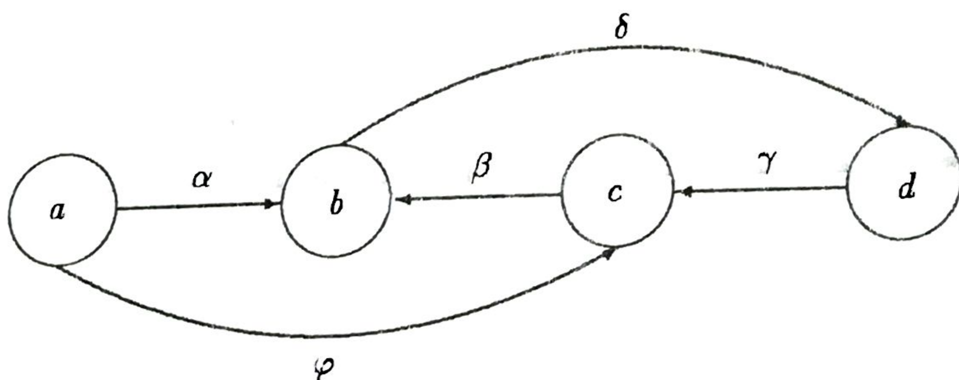
Нека $v_1, v_2, \ldots, v_n, v_{n+1}$ е произволна последователност от върхове. *Верига* ще наричаме всяка последователност от дъги $\alpha_1, \alpha_2, \ldots, \alpha_n$, за която

$$\alpha_i = (v_i, v_{i+1}) \quad \text{или} \quad \alpha_i = (v_{i+1}, v_i), \quad i = 1, 2, \ldots, n,$$

т.е. краища на дъгата α_i се явяват върховете v_i и v_{i+1}. Върхът v_1 се нарича *начало на веригата (начален връх на веригата)*, а върхът v_{n+1} се нарича *край на веригата (краен връх на веригата)*. Понякога ще казваме, че веригата съединява началния връх с крайния.

Под *дължина (мощност) на веригата* се разбира броят на участващите в нея дъги, ако приемем, че всички дъги са с дължина единица. Впоследствие за дъгите ще въведем понятието тегло (дължина, цена) и оттам понятието тегло (дължина) на веригата.

ПРИМЕР 1.2.



В пример 1.2 дъгите α, β, γ образуват верига с дължина 3, която съединява върха a с върха d.

Верига, за която $\alpha_i = (v_i, v_{i+1})$ за всяко $i = 1, 2, ..., n$, се нарича *път (ориентирана верига)*. Понятията *дължина на път*, *начален* и *краен връх* се определят както при верига.

Например дъгите α, δ определят път с дължина 2, свързващ върха a с върха d. Ясно е, че според дадените определения всеки път е верига, но не всяка верига е път (е ориентирана). Например дъгите γ, β, α са верига, но не са път между върховете d и a.

Контур се нарича верига, на която началният и крайният връх съвпадат. Под *дължина на контура* се разбира дължината на съответната верига.
Цикъл се нарича път, на който началният и крайният връх съвпадат. Под *дължина на цикъла* се разбира дължината на съответния път.

1. Ясно е, че всеки цикъл е контур, но не всеки контур е цикъл. Например дъгите δ, γ, β от последния пример образуват цикъл с дължина 3, който в същото време е и контур. Дъгите α, β, φ образуват контур с дължина 3, който не е цикъл.

2. При изчисляване на дължината всяка дъга се отчита толкова пъти, колкото пъти тя влиза в дадения път.

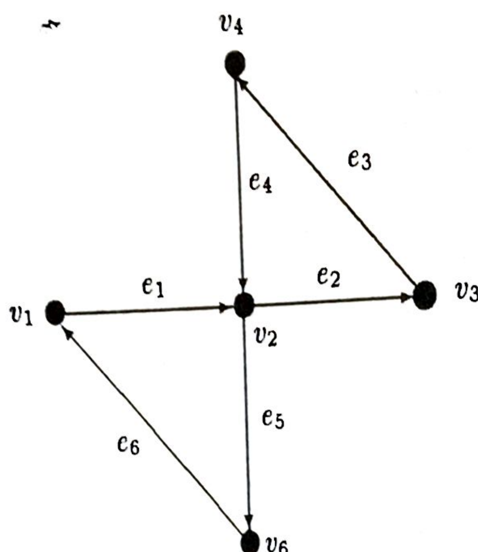
Една верига, път, контур или цикъл се наричат *прости**, ако нито един връх не е инцидентен с повече от две дъги — с други думи, ако веригата, пътят, контурът или цикълът не съдържат вътре в себе си контури.

При неориентирани графи ще използваме понятията път (прост път) и цикъл (прост цикъл), които по естествен и аналогичен начин се дефинират.

*В различните литературни източници (вж. [1]—[3]) терминологията е различна. Например в [3] при неориентирани графи основното понятие е маршрут (при нас — път). Маршрутът с различни ребра е наречен верига, а веригата с различни върхове — път, т.е. тяхното понятие път съвпада с нашето прост път. В [2] терминологията е трета и т.н.

В неориентирани графи вместо път ще употребяваме и термина маршрут.

ПРИМЕР 1.3.



В пример 1.3, дъгите e_1, e_2, e_3, e_4, e_5, e_6 образуват цикъл с начален и краен връх v_1, но този цикъл не е прост, тъй като върхът v_2 е инцидентен с повече от две дъги, т.е. вътре в този цикъл се съдържа цикъл.

Цикълът, образуван от дъгите e_2, e_3, e_4, е прост цикъл. Пътят e_1, e_2, e_3, e_4 не е прост път, тъй като съдържа в себе си цикъл. Пътят e_1, e_2, e_3 е прост път.

ЗАДАЧА 1.1. Изобразете графично проста верига и верига, която не е такава.

1.2. Видове графи. Подграф. Операции с графи. Дървета

Вече споменахме, че *прост граф* — това е граф без паралелни дъги (ребра) и примки.

Когато за всяка двойка върхове съществува ребро, инцидентно с тях, графът се нарича *пълен*. Бележи се с K_n, когато графът е неориентиран с n върха.

Понякога на всяка дъга (ребро) на графа G се приписва (съпоставя) число, наречено *"тегло"*, *"дължина"*, *"стойност"*, *"цена"* на дъгата. Приписаните на всяка дъга (ребро) числа могат да са няколко и освен на дъгите (ребрата), съответни числа могат да бъдат съпоставени и на върховете. В различните

случаи, в зависимост от естеството на задачата се употребява най-близката по смисъл дума. Например при търсене на минимални (най-кратки) пътища, теглото на всяка дъга наричаме "*дължина на дъгата*".

> Граф, чийто дъги (ребра) или върхове имат съответни тегла, се нарича *мрежа*.

Най-често теглото на дъгата (v_i, v_j) ще бележим със $c(v_i, v_j)$, c_{ij} или $c(e_k)$, където $e_k = (v_i, v_j)$, както и с $d(v_i, v_j)$, $p(v_i, v_j)$.

Ако ρ е някакъв път, например път от дъгите $e_1, e_2, \dots, e_k$, под *тегло (дължина, стойност, цена) на пътя* ρ се разбира най-често сумата от теглата (дължините, стойностите, цените) $c(e_i)$ на неговите дъги, т.е.

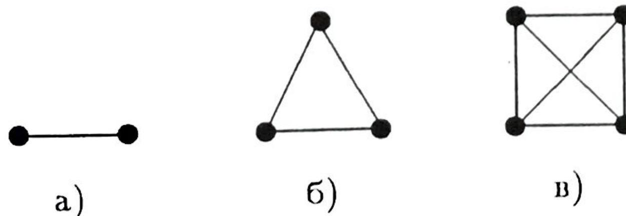
$$c(\rho) = \sum_{e_i \in \rho} c(e_i).$$

Ще припомним, че когато става въпрос за броя на участващите в даден път (или верига) дъги, ще употребяваме термина мощност на пътя. Ясно е в кои случаи дължината на пътя съвпада с мощността.

ЗАДАЧА 2.1. Вярно ли е твърдението "Един граф е свързан тогава и само тогава, когато няма изолирани върхове"?

Отг.: Не (вж. пример 2.3 б).

ПРИМЕР 2.1.



Изобразените в пример 2.1 графи са пълни, следователно и свързани.

Неориентиран граф, в който за всеки два върха съществува път, който ги свързва, се нарича *свързан граф*.

Ориентиран граф, на който съответният му неориентиран дубликат е свързан, се нарича *свързан ориентиран граф*, т.е. ориентиран граф, в който за всеки два върха съществува верига, която ги свързва.

Граф, който не съдържа цикли, се нарича *ацикличен*.

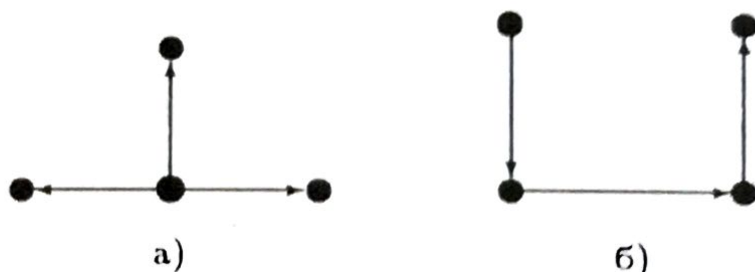
Граф, който има един връх, се нарича *тривиален*.

Графът $G = (V, A)$ се нарича *еднороден (регулярен)*, ако в него всички върхове имат еднакви степени. Пълният граф K_n е $(n-1)$ еднороден.

По-късно ще разгледаме и други видове специални графи – биполярни, ойлерови, хамилтонови и др.

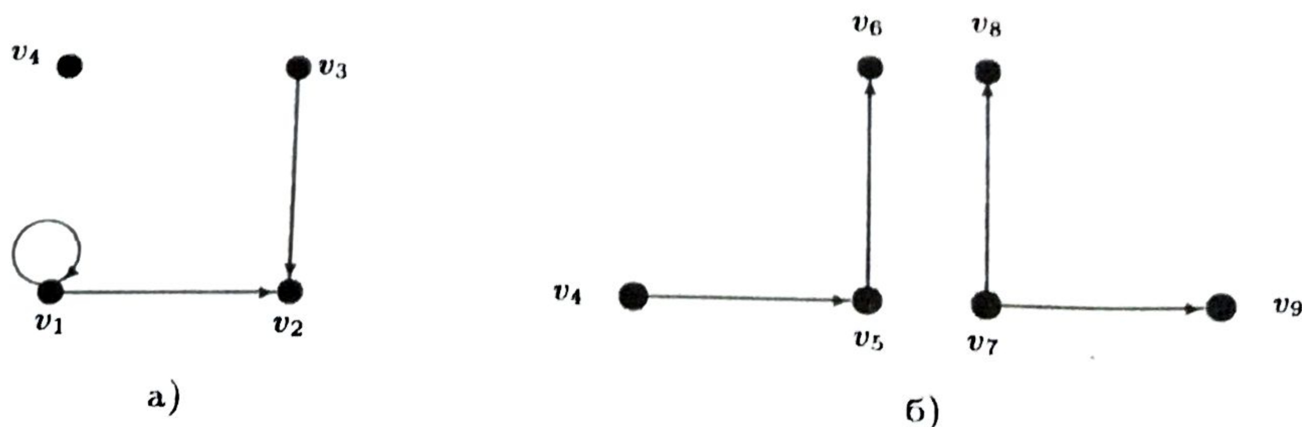
Поради очевидната връзка между ориентиран граф и двучленна релация, дефинирана в множество V, по естествен път се въвеждат понятията *рефлексивен граф* (за всеки връх има примка), *симетричен граф* (за всяка дъга съществува противоположна), *антисиметричен граф* (няма противоположни дъги и примки) и *транзитивен граф* (ако има ориентиран път от v_i до v_j, дъгата $(v_i, v_j) \in E$).

ПРИМЕР 2.2.



Изобразените в пример 2.2 графи са свързани, но не са пълни.

ПРИМЕР 2.3.



Изобразените в пример 2.3 а) и б) графи не са свързани (са несвързани), защото няма верига, свързваща върха v_4 с v_3, както няма верига, свързваща v_5 с v_9.

Всеки граф G може да се разглежда като някаква съвкупност от свързани графи. Тези графи се наричат *компоненти* на дадения граф. (Свързаният граф може да се разглежда като еднокомпонентен.)

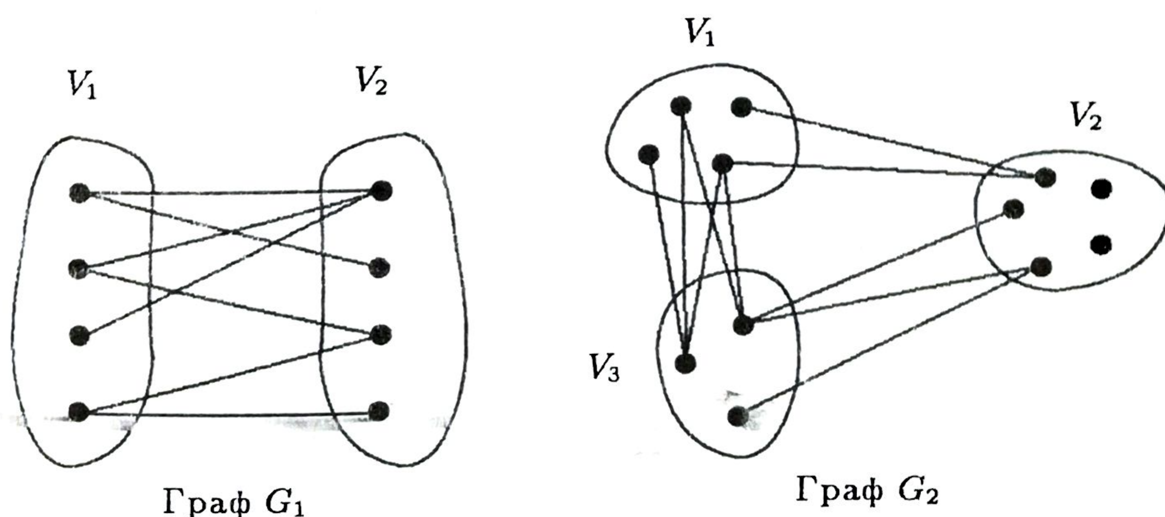
ЗАДАЧА 2.2. Кои са компонентите на графите от пример 2.3.

Графът $G = (V, A)$ се нарича *2-хроматичен (биполярен)*, ако множеството от върхоте му V може да се разбие на две подмножества V_1 и V_2, такива, че $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ и всяко ребро съединява връх от V_1 с връх от V_2 (никои два дърха от едно и също V_i не са съединени с ребро). Класовете V_1 и V_2 се наричат *хроматични класове*.

По естествен и аналогичен начин може да се дефинира r-хроматичен граф при $r > 2$ (направете го).

Биполярните графи ще бележим обикновено с $G = (V_1, V_2, A)$ или $G = (V_1 \cup V_2, A)$.

ПРИМЕР 2.4.



Граф G_1 Граф G_2

Графът G_1 от пример 2.4 е 2-хроматичен (биполярен), а графът G_2 е 3-хроматичен.

Ако в простия биполярен граф G с класове (V_1, V_2), за всеки връх $v_i \in V_1$ и $v_j \in V_2$ съществува ребро (v_i, v_j), то графът G се

нарича *пълен биполярен граф* и се означава с $K_{r,s}$, ако V_1 има r върха, а V_2 има s върха.

Аналогично се дефинира и *пълен r-хроматичен граф* (направете го). Графите от пример 2.4 не са пълни r-хроматични графи ($r = 2,\ 3$).

Биполярните графи имат някои специфични свойства, характерни само за тях. Това в много случаи се оказва полезно улеснение при решаването на редица практически проблеми, които се описват с такива графи. Ще дадем една характеризация на понятието биполярен граф.

▷ *ТЕОРЕМА 2.1. Неориентираният граф G е биполярен тогава и само тогава, когато не съдържа цикли с нечетна дължина.*

Доказателство: *Необходимост:* Нека $G = (V, A)$ е биполярен граф с класове X и Y, т.е.

$$X \cup Y = V \quad и \quad X \cap Y = \emptyset.$$

Да допуснем, че съществува цикъл с нечетна дължина $x_1, x_2, \ldots, x_q, x_1$, който стартира от връх $x_1 \in X$. Очевидно следващият връх (вторият връх) от цикъла трябва да принадлежи на Y, по-следващият (третият) отново на X и т.н. Следователно върховете с нечетни номера в цикъла принадлежат на X, а тези с четни номера — на Y. Допускането, че съществува цикъл с нечетна дължина означава, че върхът x_1, който по предположение беше от X, трябва да принадлежи на Y, което е невъзможно.

Достатъчност[2]: Нека сега в графа G не съществува цикъл с нечетна дължина. Да си изберем върха x_i и да го маркираме с "+". Да изпълним следната итерационна процедура.

Взимаме вече маркирания връх x_i и маркираме всички върхове от множеството $\Gamma(x_i)$ със знак, противоположен на този, с който е маркиран x_i. Изпълняваме тази операция, докато или

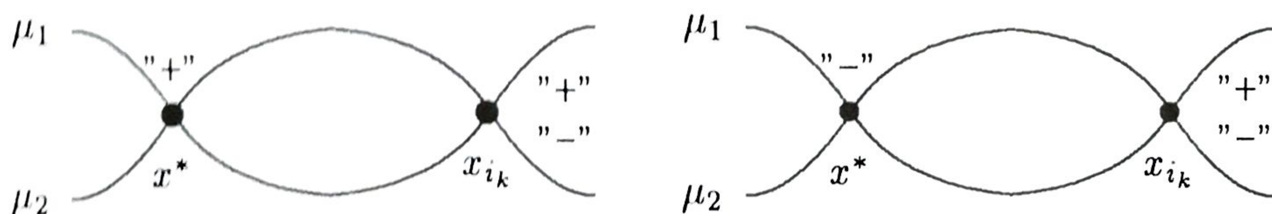
(*) всички върхове са маркирани и то така, че всеки два съседни върха са маркирани с противоположни знаци, или

(**) някой връх, например x_{i_k}, който вече е маркиран с един от знаците, може да бъде маркиран от друг връх със знак противоположен на този, който вече има, или

(***) за всеки маркиран връх x_i, всички върхове от множеството $\Gamma(x_i)$ вече са маркирани, но съществуват все още немаркирани върхове.

В случая (*) всички върхове, отбелязани с "+" отнасяме към множеството X, а отбелязаните (маркираните) с "−" към множеството Y. Очевидно в този случай графът е биполярен.

В случая (**) върхът x_{i_k} трябва да бъде маркиран с "+" в някой път (маршрут) μ_1, състоящ се от върховете x_{i_1}, x_{i_2}, ..., x_{i_k}, като знаците "+" и "−", съпоставяни на тези върхове при движение по маршрута μ_1 (от върха x_{i_1} към върха x_{i_k}) образуват редуваща се последователност от плюсове и минуси (или обратно). Аналогично върхът x_{i_k} ще бъде маркиран с "−" в някой маршрут μ_2. Нека x^* е предпоследния (последен е x_{i_k}) общ връх на маршрутите μ_1 и μ_2, както е показано на чертежа долу.



Ако върхът x^* е маркиран с " + ", то участъкът от x^* до x_{i_k} в μ_1 ще бъде четен, а участъкът от x^* до x_{i_k} в μ_2 ще бъде нечетен. Ако върхът x^* е маркиран с " − ", то участъкът в маршрута μ_1 ще бъде нечетен, а в μ_2 — четен. Следователно, цикълът, състоящ се от участъка $(x^* - x_{i_k})$ на μ_1 и съответния участък $(x_{i_k} - x^*)$ на μ_2 има нечетна дължина. Това противоречи на предположението, че графът G не съдържа цикли с нечетна дължина, т.е. случай $(**)$ е невъзможен.

Случай $(* * *)$ означава, че между маркиран и немаркиран връх не съществува ребро. Тогава графът се разбива на две или повече части, всяка от които може да се разглежда отделно. С други думи, отново ще стигнем до случай $(*)$, което доказва верността на теоремата. ◁

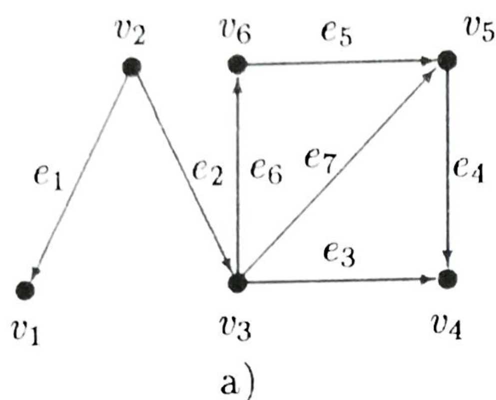
Ще дадем няколко определения в неориентиран граф, които по лесен и естествен начин се въвеждат и за ориентирани графи.

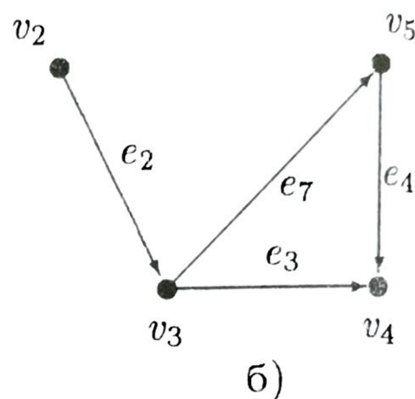
Нека $G = (V, A)$ е граф. Графът $G' = (V', A')$ се нарича *подграф* на G, ако V' и A' се явяват съответно такива подмножества на V и A, че реброто (v_i, v_j) принадлежи на A', само когато v_i и v_j принадлежат на V'.

Графът G' е *собствен подграф* на G, ако множествата V' и A' са собствени подмножества съответно на V и A.

Понякога, това, че G' е подграф на графа G ще бележим с $G' \subseteq G$.

Нека $G = (V, A)$ е произволен граф. Нека $V' \subseteq V$ и A' е множеството ребра на графа, които са инцидентни само с върхове от V'. Графът $G' = (V', A')$ се нарича *подграф на графа G, породен от върховете V'*.





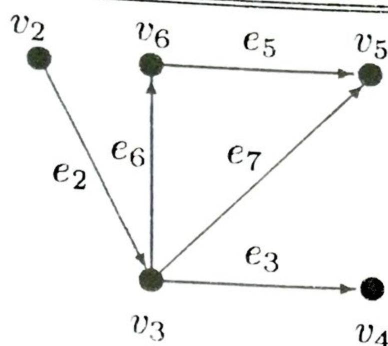
Черт. 1.2

На черт. 1.2 а) и б) са изобразени съответно граф G и подграфа G', породен от върховете $V' = \{v_2, v_3, v_4, v_5\}$.

По естествен път се дефинира и понятието подграф, породен от някакво множество от ребра.

Нека $G = (V, A)$ е граф. Нека $A' \subseteq A$ и V' е множеството от върхове на графа, инцидентни с ребрата A'. Графът $G' = (V', A')$ се нарича *подграф на G, породен от ребрата A'*.

На черт. 1.3 е изобразен подграфа, породен от множеството дъги $E' = \{e_2, e_3, e_5, e_6, e_7\}$ на графа от черт. 1.2 а).



Черт. 1.3

Подграфът G' се нарича *максимален подграф* на G по отношение на някакво свойство $P(roperty)$, ако G' притежава това свойство и не се явява собствен подграф на никой друг подграф на G, притежаващ това свойство.

По аналогичен и естествен начин може да се дефинира и понятието *минимален подграф* на графа G по отношение на свойство P.

Аналогично се определя и понятието *максимално и минимално подмножество* на някакво множество по отношение на свойството P.

От казаното е ясно, че множеството върхове V' на ребрено пороения подграф се явява минимално подмножество на V, съдържащо крайните върхове на ребрата от A'.

Очевидно пък множеството ребра A', на породения от върховете V' подграф, се явява максимално подмножество на A, чийто крайни върхове са от V'.

Компонентите на графа G се явяват максимални свързани подграфи на графа G, а покриващото дърво на един свързан граф G (което ще дефинираме малко по-късно) се явява минимален свързан покриващ подграф на графа G.

Ако $G = (V, A)$ е прост граф, графът $\overline{G} = (V, A')$ ще наричаме *допълнение на графа* G, ако реброто $(v_i, v_j) \in A'$, точно когато $(v_i, v_j) \notin A$, т.е. два върха са съседни в допълнението $\overline{G}$ тогава и само тогава, когато не са съседни в G.

Ще разгледаме някои основни операции (бинарни и унарни) с графи.

Обединение на графи. Ако $G_1 = (V_1, A_1)$ и $G_2 = (V_2, A_2)$, то обединение на графите G_1 и G_2 ще наричаме графа

$$G_1 \cup G_2 = (V_1 \cup V_2,\ A_1 \cup A_2).$$

Дефиницията може да се обобщи за произволен брой графи.

Сечение на графи. Ако $G_1 = (V_1, A_1)$ и $G_2 = (V_2, A_2)$, то сечение на графите G_1 и G_2 ще наричаме графа

$$G_1 \cap G_2 = (V_1 \cap V_2,\ A_1 \cap A_2).$$

Често се използва и унарната операция *отстраняване на връх* — графът $G - v_i$ е породеният от върховете $V - v_i$ подграф на графа G, т.е. $G - v_i$ е графът, който се получава от G след отстраняване на върха v_i и инцидентните с него ребра.

Използва се и операцията *отстраняване на ребро* — графът $G - e$ се получава от G след отстраняване на реброто e (крайните върхове на реброто не се отстраняват от G).

Отстраняването на връх и ребро по естествен начин се обобщава за произволен брой върхове и ребра.

Отъждествяване на два върха v_i и v_j в графа G се дефинира като замяна на тези върхове с нов връх, така че всички ребра на графа G, инцидентни с v_i и v_j, стават инцидентни с новия връх.

Свиване. Това е операция, при която се отстранява ребро e и се отъждествяват крайните върхове на това ребро. Тази операция се използва в някои оптимизационни алгоритми, както ще видим по-късно.

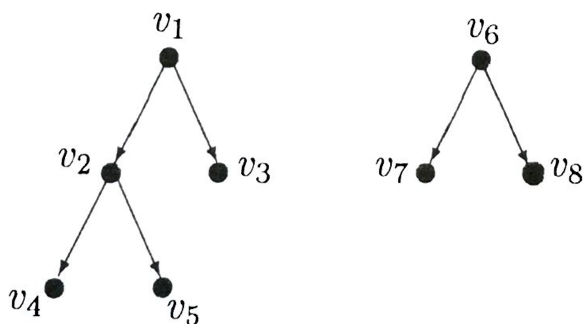
Два графа G_1 и G_2 се наричат *изоморфни*, ако съществува такова биективно изображение f между техните върхове и ребра, че:

$$\begin{matrix} v_i \xrightarrow{f} v_i' \\ v_j \xrightarrow{f} v_j' \end{matrix} \iff (v_i, v_j) \xrightarrow{f} (v_i', v_j').$$

Една съвкупност от ребра се нарича *дърво*, ако за нея са изпълнени следните две условия:
1. ребрата пораждат свързан граф;
2. няма цикли.

Ориентирано дърво е дърво, на което всеки връх с изключение на един (например v_1) има полустепен на входа единица, а върхът v_1 има полустепен на входа 0. С други думи, никои две дъги не отиват (не влизат) в един и същ връх, докато за броя на изходящите от върховете дъги няма ограничение. Единственият връх v_1, в който няма входящи (влизащи) дъги, се нарича *корен на ориентираното дърво.*

На черт. 1.5 а) е изобразено ориентирано дърво с корен v_2. Съвкупността от дъги, която се състои от едно или повече дървета, се нарича *лес (гора)*.



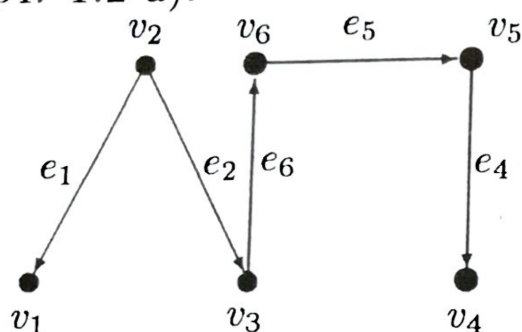
Черт. 1.4

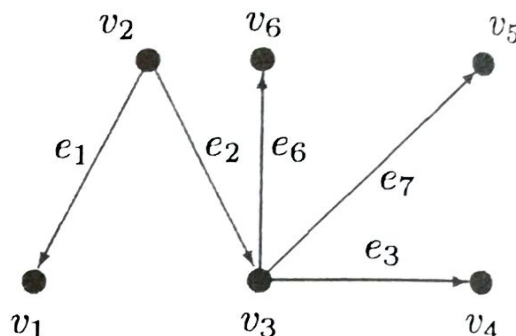
Съвкупността от дъги, представена на черт. 1.4, образува лес (гора) от две дървета.

Нека $G = (V, A)$ е произволен граф. Всяко дърво, образувано от ребра на G, включващо всички върхове на графа, се нарича *покриващо дърво*. Аналогично се дефинира понятието *покриващо ориентирано дърво.*

По-късно ще докажем, че за всеки свързан неориентиран граф има покриващо дърво, докато граф с повече компоненти (вж. черт. 1.4) няма покриващо дърво.

На черт. 1.5 са изобразени покриващи дървета за графа от черт. 1.2 а).





а) б)

Черт. 1.5

Вярно ли е, че във всеки свързан ориентиран граф има покриващо ориентирано дърво?

Очевидно дърво, състоящо се от едно ребро, включва два върха, от две ребра — три върха и т.н. (индуктивно), дърво от $n-1$ ребра включва n върха. От това следва, че всяко покриващо дърво на един свързан граф с n върха се състои от $n-1$ ребра (вж. черт. 1.5).

По естествен път се дефинират и следните понятия:

- *ориентиран лес (гора)* — лес (гора), състоящ се от ориентирани дървета;
- *покриващо ориентирано дърво* — ориентирано дърво, което е и покриващо в същото време;
- *покриващ ориентиран лес* - ориентиран лес, който включва всички върхове на графа.

Дърво в графа G ще наричаме всеки свързан ацикличен подграф на графа G.

Ако T е покриващо дърво в графа $G = (V, A)$, то подграфът T^* на G, съдържащ всички негови върхове и само ребрата, неучастващи в T, се нарича *ко-дърво* на T. Очевидно, ко-дървото T^* не е задължително да бъде свързано. Ребрата на едно покриващо дърво T се наричат *клони* на дървото, а ребрата на съответното му ко-дърво T^* се наричат *хорди*.

Ще дадем още две важни характеристики на един граф G с n върха и m ребра (дъги), съдържащ k компоненти.

Рангът на графа G се обозначава чрез $r(G)$ и се определя като $n - k$, т.е. $r(G) = n - k$.

Цикломатичното число на графа G се означава с $\mu(G)$ и се дефинира като $m - n + k$.

Очевидно е равенството $r(G) + \mu(G) = m$.

Тези две важни характеристики са свързани с размерността на подпространствата на циклите и разрезите в един граф и се използват широко в така нареченото алгебрично направление в теория на графите. Не е трудно да се съобрази, че цикломатичното число на произволен граф G е неотрицателно. Цикломатичното число на всяко дърво е нула. Дайте пример за граф G, на който цикломатичното число $m - n + k = 0$, но G не е дърво. Колко е цикломатичното число на произволен прост цикъл?

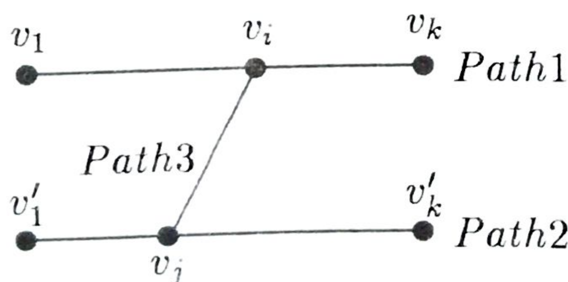
1.3. Свързаност. Покриващи дървета. Разрези. Цикли. Силна свързаност

1. Свързаност

Ще изложим някои свойства на свързаните графи.

▷ *ТЕОРЕМА 3.1. Нека $G = (V, A)$ е произволен свързан граф. Всеки два прости пътя с максимална дължина (максимален брой ребра) имат общ връх.*

Доказателство: Нека $Path1$ и $Path2$ са два прости пътя с максимална дължина (в простия път няма повтарящи се върхове). Да допуснем, че те нямат общ връх, както е показано на черт. 1.6.



Черт. 1.6

Поради това, че графът G е свързан, между всеки два негови върха ще съществува прост път. Да означим с $Path3$ пътя между върховете $v_i \in Path1$ и $v_j \in Path2$. Точно един участък от пътя $Path1$, например участъкът $\{v_1, v_i\}$ има дължина $\geq \frac{1}{2}|Path1|$. Аналогично и без ограничение на общността на разсъждения, участъкът $\{v_j, v'_k\}$ има дължина $\geq \frac{1}{2}|Path2|$. Но $|Path1| = |Path2| = Max$, откъдето

$$|\{v_1, v_i\} \cup Path3 \cup \{v_j, v'_k\}| \geq \frac{1}{2}Max + \frac{1}{2}Max + |Path3| = Max + |Path3|,$$

което означава, че намерихме път с дължина по-голяма от максималната. Противоречие. С това теоремата е доказана. ◁

Очевидно е вярно и следното твърдение.

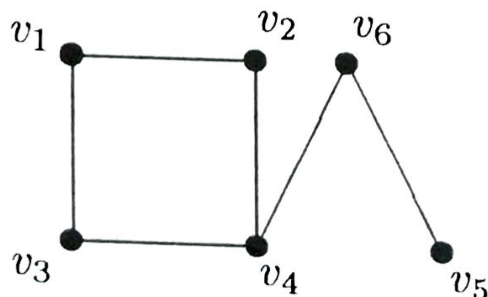
▷ *ТЕОРЕМА 3.2. Ако графът $G = (V, A)$ е свързан, то графът, който се получава от G след отстраняване на циклично ребро e, т.е. графът $(V, A - e)$ също е свързан (циклично ребро е ребро, участващо в цикъл).* ◁

Ще дадем още едно важно понятие от теория на графите.

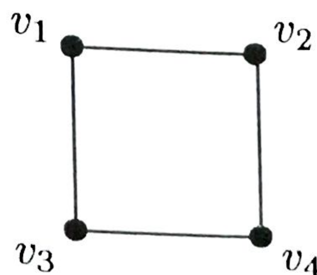
	Свързваща точка. Върхът v_i в графа G се нарича свързваща точка, ако графът $G - v_i$ има по-голям брой компоненти в сравнение с G.

Очевидно изолираните върхове, съгласно даденото определение, не могат да бъдат свързващи точки в графа. Също така тривиалният граф (имащ един връх) няма свързващи точки.

Свързани графи, които нямат свърващи точки, се наричат *неразделими*, а тези, които имат — *разделими*. Графът на черт. 1.7 а) е разделим. Той има две свързващи точки — v_4 и v_6.



а)

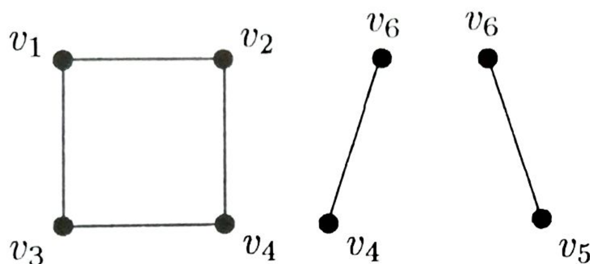


б)

Черт. 1.7

Напомняме, че при операцията отстраняване на връх от граф, се отстраняват и инцидентните с него ребра. Графът на черт. 1.7 б) е неразделим. (Несвързаните графи се считат за разделими.)

Всеки максимален неразделим подграф на разделимия граф G се нарича *блок*. На черт. 1.8 са дадени блоковете на разделимия граф от черт. 1.7 а).



Черт. 1.8

Ще дадем още една еквивалентна дефиниция на понятието свързваща точка, изясняваща допълнително същността на това понятие.

▷ *ТЕОРЕМА 3.3. Нека G е свързан граф. Върхът v е свързваща точка тогава и само тогава, когато съществуват два върха v_i и v_j, различни от v, такива че всеки $(v_i - v_j)$ път съдържа върха v.*

Доказателство:
Достатъчност: Достатъчността е очевидна поради това, че ако всеки $(v_i - v_j)$ път съдържа v, графът $G - v$ няма да бъде свързан, т.е. точката v ще бъде свързваща.

Необходимост: Да допуснем, че v е свързваща точка, но за всеки два върха v_i и v_j от G различни от v, съществува път несъдържащ v. Тъй като по условие v е свързваща точка, графът $G - v$ ще бъде поне двукомпонентен. От допускането следва, че за върховете v_i и v_j, принадлежащи на различни компоненти на $G - v$, ще съществува $(v_i - v_j)$ път, което е невъзможно. Следователно всеки $(v_i - v_j)$ път съдържа върха v. ◁

ЗАДАЧА 3.1. Да се докаже, че необходимо и достатъчно условие свързаният граф G да бъде разделим, е да съществува връх v, който е единствен общ връх за два собствени нетривиални подграфи G_1 и G_2, чието обединение дава G.

Упътване: Виж теорема 3.3.

Не е възможно всички върхове на един разделим граф G да бъдат свързващи точки. Ще изясним това, като докажем следната теорема.

▷ *ТЕОРЕМА 3.4. Всеки нетривиален свързан граф съдържа поне два върха, които не са свързващи точки.*

Доказателство: Твърдението очевидно е вярно за произволен свързан граф с два върха. Да допуснем, че твърдението е вярно за всеки нетривиален свързан граф с не повече от $n-1$ върха ($n > 2$). Да разгледаме свързания граф G с n върха:

а) Ако G няма свързващи точки, твърдението е доказано;

б) Нека в графа G v е свързваща точка и G_1, G_2, ..., G_s са компонентите на графа $G - v$.

1) Ако някоя от компонентите G_i е тривиален граф, неговият единствен връх не е свързваща точка.

2) Нека G_i е произволна нетривиална компонентна. Според индуктивното допускане направено по-горе, в G_i ще съществуват поне два върха v' и v'', които не са свързващи точки.

- ако един от върховете v' и v'' не е съседен с върха v в графа G, очевидно този връх не е свързваща точка в G;

- ако върховете v' и v'' са съседни с v в G, те няма да бъдат свързващи точки в G.

От 1) и 2) следва, че всяка компонента на $G - v$ съдържа поне един връх, който не е свързваща точка в G. По индукция следва, че в графа G поне два върха не са свързващи точки. ◁

2. Дървета

▷ *ТЕОРЕМА 3.5.* [3] *Нека G е граф с n върха и m ребра. Тогава следните пет твърдения са еквивалентни (дефиниции за дърво):*

а) G е свързан ацикличен граф, т.е. G е дърво;

б) G е граф, в който за всеки два върха съществува единствен прост път (път без повтарящи се върхове), свързващ тези върхове;

в) G е свързан граф с $n-1$ ребра, т.е. $m = n-1$;

г) G е ацикличен граф, за който $m = n-1$;

д) G е ацикличен граф и при съединяването на два негови произволни несъседни върха с ребро се получава граф с точно един прост цикъл.

Доказателство: "а) $\Longrightarrow$ б)":

Ако допуснем, че има два прости пътя в графа G, съединяващи върховете v_i, v_j, следва съществуването на цикъл, което противоречи на а).

"б) $\Longrightarrow$ в)":

Щом съществува точно един прост път между всеки два върха, следва че графът G е свързан. Остава да покажем, че за този свързан граф G броят на ребрата е $n-1$.

Ако графът G съдържа 1, 2 или 3 върха, твърдението очевидно е вярно. Да допуснем, че твърдението е вярно за граф с $n-1$ върха. Ще докажем, че твърдението е вярно и за граф с n върха.

Да разгледаме графа $G-e$, където e е произволно ребро (отстраняваме реброто e без да отстраняваме върхове от графа). Тъй като по условие реброто e е единственият прост път между крайните му върхове, графът $G-e$ няма да бъде свързан и освен това ще бъде двукомпонентен (в противен случай графът G няма да бъде свързан). Да означим с G_1 и G_2 компонентите на графа $G-e$. Нека n_1 и n_2 са върховете на компонентите G_1 и G_2, а m_1 и m_2 са съответно броя на ребрата на тези компоненти. Според индуктивното допускане за броя на ребрата на G_1 и G_2 ще имаме:

$$m_1 = n_1 - 1, \quad m_2 = n_2 - 1.$$

От друга страна очевидно

$$m = m_1 + m_2 + 1 \quad \text{и} \quad n = n_1 + n_2.$$

Следователно $m = n_1 - 1 + n_2 - 1 + 1 = n - 1$. Оттук по индукция следва верността на твърдението.

"в) $\Longrightarrow$ г)":

Да допуснем, че графът G е свързан, има $n-1$ ребра, но в него съществува един прост цикъл. Очевидно отстраняването на едно ребро e от този цикъл (*циклично ребро*) няма да наруши свързаността в получения граф $G_1 = G - e$, т.е. графът G_1 ще бъде свързан и ацикличен, т.е. ще бъде дърво. (Ако допуснете съществуването на няколко прости цикли, след неколкократно отстраняване на циклични ребра ще стигнете до граф G_s, който е дърво.) Следователно броят на неговите ребра ще бъде, както вече доказахме $n-1$. Тъй като $G_1 = G - e$, броят на неговите ребра ще бъде от друга страна $(n-1)-1$, т.е.

$$(n-1) - 1 = n - 1 \implies -1 = 0,$$

което е невъзможно, поради това отхвърляме допускането, т.е. графът G е ацикличен. (В по-общия случай, за дървото G_s ще имаме $m - s = n - 1 \Longrightarrow (n-1) - s = n - 1 \Longrightarrow s = 0$, т.е. G е ацикличен.)

"г) $\Longrightarrow$ д)":

Да означим с $G_1, G_2, \ldots, G_s$ компонентите на графа G. Нека n_i и m_i са съответно върховете и ребрата в компонентите G_i, $1 \leq i \leq s$. Тъй като G_i са компоненти, те по дефиниция са свързани и освен това поради ацикличността на графа G са ациклични, т.е. всяка компонента G_i е дърво, следователно за всяко $1 \leq i \leq s$, $m_i = n_i - 1$. Тогава

$$m = \sum_{i=1}^{s} m_i = \sum_{i=1}^{s} (n_i - 1) = n - s.$$

По условие $m = n - 1$, следователно

$$n - 1 = n - s \implies s = 1,$$

т.е. графът G е еднокомпонентен и ацикличен, т.е. е дърво и както вече доказахме, в него ще съществува единствен прост път между всеки два върха v_1 и v_2. Очевидно добавянето на реброто (v_1, v_2) към графа G ще образува с единствения прост път между v_1 и v_2 точно един прост цикъл.

"д) $\implies$ а)":

По предположение графът G е ацикличен, остава да покажем, че е свързан. Да допуснем противното, т.е. графът G е поне двукомпонентен. Тогава за всеки два върха v_i и v_j, принадлежащи съответно на различни компоненти, няма да съществува път, който ги свързва. Следователно добавянето на реброто (v_i, v_j) към графа G няма да води до образуването на цикъл, което обаче противоречи на предположението д). Следователно графът G е еднокомпонентен (свързан) и по предположение е ацикличен, следователно графът G е дърво.

С това доказателството на теоремата е направено. ◁

СЛЕДСТВИЕ 1. Нека G е граф с n върха и $G_1 \subseteq G$ с n върха и m_1 ребра. Следните твърдения са еквивалентни:

а) G_1 е покриващо дърво за G;

б) между всеки два върха в G_1 съществува единствен прост път;

в) G_1 е свързан и $m_1 = n - 1$;

г) G_1 е ацикличен и $m_1 = n - 1$;

д) ако G_1 е ацикличен и съединим два произволни негови несъседни върха с ребро, то полученият граф има точно един прост цикъл.

СЛЕДСТВИЕ 2. Нека G е граф с n върха и $G_1 \subseteq G$. Подграфът G_1 е покриващо дърво за G тогава и само тогава, когато G_1 е ацикличен, свързан граф с $n - 1$ ребра.

В резюме можем да кажем следното:

Нека G е граф с n върха и $G_1 \subseteq G$. Ако подграфът G_1 притежава три от следните четири свойства: а) има n върха; б) свързан е; в) има $n-1$ ребра; г) ацикличен е, то G_1 ще бъде покриващо дърво за графа G.

С изключение на двойката условия в) и г) никои други две от горните четири условия не са достатъчни, за да бъде подграфът G_1 на свързания граф G покриващо дърво за G.

▷ *ТЕОРЕМА 3.6. Нека G е граф с n върха. Подграфът G' е покриващо дърво за графа G тогава и само тогава, когато G' е ацикличен и има $n-1$ ребра.*

Доказателство: *Необходимост:* Необходимоста е очевидно следствие от доказаната вече теорема 3.5.

Достатъчност: Тъй като по предположение G' е ацикличен, достатъчно е да докажем, че G' е свързан граф с n върха. Да означим с G_1, G_2, ..., G_s компонентите на G', с n_1, n_2, ..., n_s броят на върховете на съответните компоненти и с n' — броят на върховете на G'.

Ясно е, че компонентите са ациклични графи, поради ацикличността на G' и тъй като са в същото време свързани, следва че всяка от компонентите е дърво. Следователно ребрата във всяка компонента ще бъдат $n_i - 1$, $1 \le i \le s$. Тъй като по условие броят на ребрата на G' е $n-1$, ще имаме

$$\sum_{i=1}^{s}(n_i - 1) = n - 1 \implies \sum_{i=1}^{s} n_i - s = n - 1,$$

т.е.

$$n' - s = n - 1.$$

Поради $n' \le n$ и $s \ge 1$, горното равенство е възможно тогава и само тогава, когато $n' = n$ и $s = 1$. Следователно графът G' е еднокомпонентен (свързан) с n върха, а по условие той е

ацикличен, което е достатъчно, за да твърдим, че той се явява покриващо дърво за графа G. ◁

Очевидно за несвързаните графи не съществува покриващо дърво. Ясно е и това, че ако в графа G има покриващо дърво T, то графът G ще бъде свързан, тъй като ще съдържа свързан подграф, включващ всички негови върхове. По естествен път възниква въпросът дали обратното е вярно. За целта ще формулираме

▷ *ТЕОРЕМА 3.7. Ако графът G е свързан, за него съществува покриващо дърво.*

Доказателство: Ако свързаният граф G е и ацикличен, той се явява покриващо дърво за себе си. Нека сега свързаният граф G не е ацикличен. Тогава очевидно графът $G_1 = G - e_1$, където e_1 е циклично ребро, ще бъде свързан и ще съдържа всички върхове на G (при отстраняването на ребро от граф не се отстраняват крайните му върхове). За графа G_1 отново имаме две възможности — да бъде цикличен или ацикличен. Ако G_1 не е ацикличен, ще повтаряме операцията отстраняване на циклично ребро, докато не стигнем до ацикличен граф G_s, който ще бъде свързан (отстраняването на циклично ребро не нарушава свързаността) и ще съдържа всички върхове на G. Очевидно графът G_s ще бъде покриващо дърво за графа G. ◁

Очевидно всеки подграф на покриващото дърво T е ацикличен подграф на G. Възниква въпросът дали всеки ацикличен подграф на G се явява подграф и на някое покриващо дърво T. Отговорът на този въпрос се съдържа в следващата теорема.

▷ *ТЕОРЕМА 3.8. Ако подграфът G' на свързания граф G е ацикличен, то G' се явява подграф на някое покриващо дърво T.*

Доказателство: Нека T е покриващо дърво на графа G.

Да разгледаме графа $G_1 = T \cup G'$. Очевидно G_1 е свързан граф, който съдържа всички върхове на графа G. Ако G_1 е и ацикличен, следва че той е покриващо дърво за графа G, а от начина по който дефинирахме G_1 е очевидно, че G' е негов подграф.

Да допуснем, че графът G_1 съдържа цикъл ρ_1. Поне едно ребро от цикъла ρ_1 не се съдържа в G' (ако всички ребра на

цикъла ρ_1 са в G', ще следва, че G' е цикличен, а той по условие не е такъв). Да отстраним от G_1 цикличното ребро e_1, което не се съдържа в G'.

Полученият по този начин граф $G_2 = G_1 - e_1$ ще остава свързан, ще включва всички върхове на G_1 (и на G) и G' ще бъде негов подграф. Ако G_2 е и ацикличен, той ще се явява покриващото дърво, на което G' ще се явява подграф.

В противен случай, с аналогични отстранявания на циклични ребра, ще стигнем до покриващо дърво, на което G' ще бъде подграф. ◁

Следващата теорема дава оценка за броя на висящите върхове (върхове със степен 1) в едно дърво.

▷ *ТЕОРЕМА 3.9. Ако T е дърво с $n > 1$ върха, съществуват поне два висящи върха.*

Доказателство: Както вече доказахме, дървото T ще съдържа $n-1$ ребра. Тогава съгласно теорема 1.1 а)

$$d(v_1) + d(v_2) + \ldots + d(v_n) = 2(n-1) \implies$$

$$\sum_{i=1}^{n} d(v_i) = 2n - 2.$$

Понеже T е дърво, всички върхове ще бъдат от степен по-голяма или равна на 1 (няма изолирани върхове). Да означим с s броя на висящите върхове, които имат степен 1. Останалите $n-s$ върха ще имат степен по-голяма или равна на 2. Тогава

$$2n - 2 = \sum_{i=s+1}^{n} d(v_i) + s \geq (n-s).2 + s \implies$$

$$2n - 2 \geq 2n - s \implies s \geq 2.$$

◁

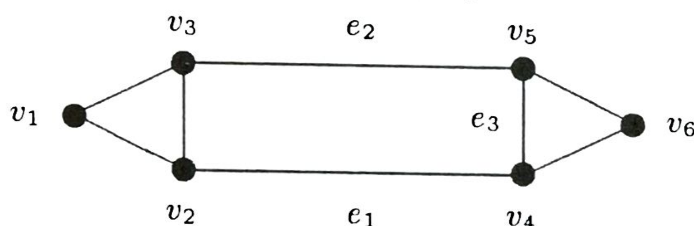
3. Разрези и цикли

Нека $G = (V, A)$ е произволен свързан граф. Множеството от ребра на този граф, чието отстраняване води до увеличаване броя на компонентите на графа (т.е. премахва свързаността), ще наричаме *разрязващо множество*.

Когато не съществува истинско подмножество на това множество от ребра със същото свойство, разрязващото множество се нарича *просто разрязващо множество*.

Може да се даде следната еквивалентна дефиниция за просто разрязващо множество — минималният брой ребра, чието отстраняване от графа G разбива графа на две компоненти G_1 и G_2.

ПРИМЕР 3.1. Да разгледаме следния граф.



Очевидно множеството ребра $\{e_1,\ e_2,\ e_3\}$ е разрязващо, тъй като отстраняването на тези ребра води до загуба на свързаност.

Това множество обаче не е просто разрязващо множество, тъй като съществува негово подмножество със същото свойство — например множеството $\{e_1,\ e_2\}$, което е просто разрязващо множество.

Не е трудно да се съобрази, че всеки нетривиален свързан граф G съдържа разрязващо и просто разрязващо множество.

▷ *ТЕОРЕМА 3.10. Дадените (по-горе) две дефиниции за просто разрязващо множество са еквивалентни.*

Доказателство: Ако S е просто разрязващо множество според втората дефиниция, т.е. S включва минимален брой ребра, за които $G-S$ се състои от точно две компоненти G_1 и G_2, то очевидно графът $G-S$ не е свързан, т.е. S е просто разрязващо множество според първата дефиниция.

Обърнете внимание, че според втората дефиниция, за ранговете на графите G и $G-S$ е изпълнено

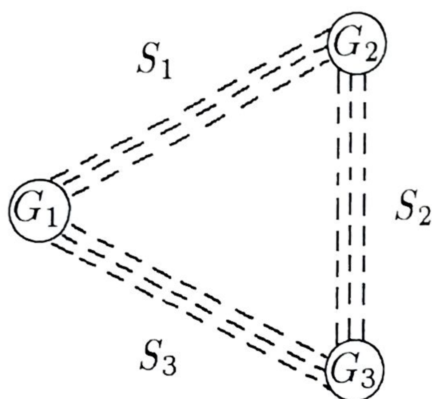
$$(*) \qquad rank\ (G) - rank\ (G - S) = 1.$$

От първата дефиниция за просто разрязващо множество следва

$$(**) \qquad rank\ (G) - rank\ (G - S) \geq 1.$$

Очевидно от $(*)$ следва $(**)$, както вече уточнихме.

Нека сега S е просто разрязващо множество според дефиниция 1, т.е. S е минимален брой ребра, за които $G - S$ не е свързан, т.е. е поне двукомпонентен. Ще докажем, че $G - S$ е двукомпонентен. Да допуснем, че $G - S$ е трикомпонентен (това не ограничава общността на разсъжденията). Да означим с G_1, G_2 и G_3 компонентите на $G - S$, както е показано на черт. 1.9,



Черт. 1.9

а с S_1, S_2 и S_3 — ребрата, чиито краища са в съответните компоненти, както е показано на чертежа, т.е. $S = S_1 \cup S_2 \cup S_3$.

Поради свързаността на G и допускането за трикомпонентност на $G - S$, най-много едно от трите множества S_i е $\emptyset$ и две по две множествата S_i са непресичащи се. Нека $S_3 \neq \emptyset$. Тогава очевидно $S_1 \cup S_2$ ще бъде истинско подмножество на S, за което $G - (S_1 \cup S_2)$ няма да бъде свързан. Това противоречи на минималността на броя ребра в S. ◁

Понятието *разрез* е тясно свързано с понятието просто разрязващо множество. Нека $G = (V, A)$ е произволен граф. Нека за подмножествата от върхове V' и V'' е изпълнено $V' \cup V'' = V$ и $V' \cap V'' = \emptyset$. Множеството от всички ребра, на които единият връх е във V', а другият — във V'', се нарича *разрез на графа* G. Разрезът обикновено се бележи с $\langle V', V'' \rangle$.

За графа от пример 3.1, ако $V' = \{v_1,\ v_2,\ v_3\}$ и $V'' = \{v_4,\ v_5,\ v_6\}$, то разрезът $\langle V', V'' \rangle$ е множеството от ребра $\{e_1,\ e_2\}$.

Ще отбележим, че разрезът $\langle V', V'' \rangle$ на графа G се явява минималния брой ребра, отстраняването на които го разбива на

графи G_1 и G_2, явяващи се подграфи, породени съответно от V' и V''. Графите G_1 и G_2 могат да не са свързани. Ако обаче тези графи са свързани, то съгласно определенията, разрезът $\langle V', V''\rangle$ ще бъде просто разрязващо множество. Ако за простото разрязващо множество S на графа G компонентите G_1 и G_2 на $G - S$ имат съответно върхове V' и V'', то простото разрязващо множество S е разрезът $\langle V', V''\rangle$.

Направените разсъждения и някои следствия от тях ще резюмираме в следните три твърдения.

▷ *ТЕОРЕМА 3.11.* [3] *(1). Разрезът $\langle V', V''\rangle$ на свързания граф G е просто разрязващо множество на графа G, ако съответните подграфи на G, породени от върховете V' и V'', са свързани графи.*
(2). Ако S е просто разрязващо множество в свързания граф G, а V' и V'' са върховете на компонентите на $G - S$, то $S = \langle V', V''\rangle$.
(3). Всеки разрез $\langle V', V''\rangle$ на свързания граф G е обединение на t "ребрено-непресичащи" се прости разрязващи множества на G, където $t \geq 1$. ◁

Докажете верността на (3) от теорема 3.11, като използвате това, че всеки разрез $\langle V', V''\rangle$ в свързан граф съдържа просто разрязващо множество, тъй като отстраняването на ребрата $\langle V', V''\rangle$ от графа G го превръща в несвързан.

▷ *ТЕОРЕМА 3.12. Нека G е свързан граф. Множеството от всички ребра, инцидентни с върха v е просто разрязващо множество тогава и само тогава, когато v не е свързваща точка.*

Доказателство: Понятието свързваща точка е дадено след теорема 3.2 и в теорема 3.3. Инцидентните ребра с върха v образуват разрез $\langle v, V - v\rangle$, тъй като тяхното отстраняване очевидно води до несвързаност и до разбиване на графа G на два подграфа G_1 и G_2, породени съответно от v и $V - v$. Тривиалният подграф G_1 очевидно е свързан.

Необходимост: Нека инцидентните с върха v ребра са просто разрязващо множество, т.е. тяхното отстраняване разбива G на две компоненти. С други думи G_1 и G_2 са свързани графи.

Щом G_2 е свързан граф, за всеки два негови върха (а това са върхове, различни от v) ще съществува път, който не съдържа върха v. Съгласно теорема 3.3 върхът v не е свързваща точка за графа G.

Достатъчност: Нека v не е свързваща точка в свързания граф G. Тогава съгласно теорема 3.3, за всеки два върха различни от v, ще съществува път между тях, който не съдържа v. Това означава, че породеният от $V - v$ подграф G_2 ще бъде свързан по определение. Съгласно (1) от теорема 3.11 разрезът $\langle v, V - v \rangle$ ще бъде просто разрязващо множество. ◁

При ориентирани графи $G = (V, E)$, понятието разрез се дефинира аналогично — множеството от всички дъги, съединяващи върхове от непразното множество V' с върхове от $V'' = V \setminus V'$.

Ориентацията на разреза $\langle V', V'' \rangle$ може да бъде избрана както от върховете на V' към върховете на V'', така и обратно — от върховете на V'' към върховете на V'. Ако е фиксирана ориентацията на разреза, можем да говорим за съвпадане (несъвпадане) на тази ориентация с ориентацията на произволна дъга от разреза.

По-късно в параграф 2.5 при разглеждането на потокови алгоритми (Теорема на Форд-Фалкерсон за максималния поток и минималния разрез), съществено ще използваме това понятие.

Ще направим една взаимна характеризация на разгледаните понятия просто разрязващо множество, покриващо дърво и цикъл, като докажем следните няколко твърдения.

ЛЕМА 3.1. Ако S е просто разрязващо множество в свързания граф G, то S съдържа поне по един клон (ребро) от всяко покриващо дърво T.

Доказателство: Нека допуснем противното, т.е. съществува покриващо дърво T на графа G, такова че S не съдържа нито едно ребро от T. Тогава очевидно графът $G - S$ ще съдържа покриващото дърво T, откъдето следва, че $G - S$ ще бъде свързан граф. Това е невъзможно, тъй като S е просто разрязващо множество. ◁

▷ *ТЕОРЕМА 3.13. Нека S е произволно множество от ребра на свързания граф G. S е просто разрязващо множество тогава и само тогава, когато S е минимално множество ребра, съдържащо поне по едно ребро от всяко покриващо дърво T на G.*

Доказателство: *Необходимост:* Ако S е просто разрязващо множество, то от лема 3.1 следва, че S съдържа поне по едно ребро от всяко покриващо дърво T. Въпросът е дали S е минимално относно това свойство.

Да допуснем, че съществува собствено подмножество S' на S със същото свойство. Тогава $G - S'$ няма да съдържа нито едно покриващо дърво T, т.е. няма да бъде свързан, но това противоречи на условието S да бъде просто разрязващо множество, т.е. минимално множество от ребра, чието отстраняване води до несвързаност.

Достатъчност: Ако S е минимално множество ребра, съдържащо поне по едно ребро от всяко покриващо дърво T на G, то очевидно $G - S$ няма да съдържа покриващи дървета и ще бъде несвързан.

Да допуснем сега, че разрязващото множество S не е просто разрязващо множество. Тогава ще съществува негово собствено подмножество S', което ще бъде просто разрязващо и съгласно доказаната вече *необходимост*, S' ще бъде минимално множество ребра, съдържащо поне по едно ребро от всяко T. Това противоречи на условието за минималност на S. ◁

ЛЕМА 3.2. Ако C е прост цикъл в свързания граф G, то той съдържа поне по едно ребро от всяко ko-дърво T^* на графа G.

Доказателство: Да припомним, че *ko*-дървото T^* на покриващото дърво T е подграф, включващ всички върхове на G и само онези ребра на G, които не са в T.

Нека C е произволен цикъл в графа G. Да допуснем, че съществува *ko*-дърво T^*, което не съдържа циклично ребро. Тогава $G-T^*$ ще съдържа цикъла C. Но $G-T^* = T$. Следователно дървото T ще съдържа цикъл, което е невъзможно. ◁

▷ *ТЕОРЕМА 3.14. Нека G е свързан граф и C е произволно множество от ребра на G. Множеството C е прост цикъл тогава и само тогава, когато то е минимално множество от ребра, съдържащо поне по едно ребро от всяко ko-дърво T^*.*

Доказателство: *Необходимост:* Нека C е прост цикъл. Съгласно лема 3.2, C ще съдържа поне по едно ребро от всяко *ko*-дърво T^*. Остава да покажем, че C е минимално множество с това свойство.

Нека C' е произволно собствено подмножество на C. Тъй като C е прост цикъл, очевидно множеството ребра C' е ацикличен подграф и съгласно теорема 3.8 ще съществува покриващо

дърво T, така че $C' \subseteq T$. Оттук *ko*-дървото T^*, нямащо общи ребра с T, няма да има и общи ребра с C', т.е. всяко собствено подмножество на C не притежава свойството да съдържа поне по едно ребро от всяко *ko*-дърво T^*. С това е доказана и минималността.

Достатъчност: Нека сега C е минимално множество ребра, съдържащо поне по едно ребро от всяко *ko*-дърво T^*.

1) Множеството C не е ациклично, иначе съгласно теорема 3.8 ще съществува покриващо дърво T, така че $C \subseteq T$. Освен това очевидно в *ko*-дървото T^* няма да има нито едно ребро от C. Това противоречи на свойствата, които притежава C по предположение. Следователно в C се съдържа поне един прост цикъл C'.

2) Ще докажем, че C е прост цикъл, т.е. всяко негово собствено подмножество вече не е такова. Да допуснем противното, т.е. съществува собствено подмножество C', което е прост цикъл. От доказаната *необходимост* следва, че C' е минимално множество ребра, съдържащо поне по едно ребро от всяко *ko*-дърво T^*. Това противоречи на направеното предположение в условието за минималност на множеството C по отношение на това свойство. ◁

Сега ще докажем една интересна и важна за теория на графите теорема, характеризираща отношението "прост цикъл — просто разрязващо множество" и изразяваща двойнствения характер на тези понятия. Като следствия от нея се доказват редица интересни резултати, даващи връзката (ортогоналност) между подпространствата на циклите и разрезите, които са свързани с напрежението и тока в електрическите вериги.

▷ *ТЕОРЕМА 3.15. Нека G е свързан граф. Всеки прост цикъл и всяко просто разрязващо множество на G имат четен брой общи ребра.*

Доказателство: Да означим със C и S съответно прост цикъл и просто разрязващо множество в графа G. Нека G_1 и G_2 са компонентите на $G - S$, които имат върхове V_1 и V_2.

1) Ако цикълът C е подграф на някоя от компонентите, то очевидно $C \cap S = \emptyset$, т.е. C и S имат 0 на брой общи ребра — теоремата е вярна.

2) Нека простият цикъл C и простото разрязващо множество S имат общи ребра. Без ограничение на общността да означим

с $v_1 \in V_1$ началото и края на цикъла. Започвайки от v_1 да обхождаме цикъла, ще стигнем до общо за цикъла и разрязващото множество ребро, което ще ни "прехвърли" в множеството V_2. Тъй като простият цикъл (ребрата му са различни) трябва да завърши в началния връх $v_1 \in V_1$, е ясно, че броят на алтернативните "прехвърляния" (ребра от $S \cap C$) е четен брой. С това теоремата е доказана. ◁

Да припомним, че с $rank\ (G)$ или $r(G)$ бележим *ранга на един граф* и $r(G) = n - k$, където n е броят на върховете, а k е броят на компонентите на графа. Числото $r(G)$ е броят на ребрата в покриващите дървета на всяка от неговите k компоненти, които са свързани.

Числото $\mu(G)$ - *цикломатичното число* определихме като

$$\mu(G) = m - r(G) = m - n + k,$$

където m е броят на ребрата на графа. Понякога цикломатичното число $\mu(G)$ се нарича *дефект*, а рангът $r(G)$ — *кощикломатично число*. В теорията на електрическите вериги горните две числа имат пряк физически смисъл.

Ако T е покриващо дърво в графа G, то очевидно добавянето на ребро (v_i, v_j) от графа G, непринадлежащо на T, към ребрата на T води до образуването на точно един прост цикъл. Този прост цикъл се състои от ребрата на T, образуващи единственият прост път между v_i и v_j и добавеното ребро. Тъй като в графа има m ребра и $n - 1$ от тях са в T, то броят на всички цикли, построени по гореописания начин, е $m - n + 1$, т.е. съвпада с цикломатичното число на G. Всички такива цикли C_1, C_2, ..., $C_{\mu(G)}$ са независими помежду си в смисъл, че всеки от тях има поне по едно ребро, непринадлежащо на никой друг цикъл. Споменатите по-горе $\mu(G)$ на брой цикли се наричат *базисни (фундаментални) цикли*

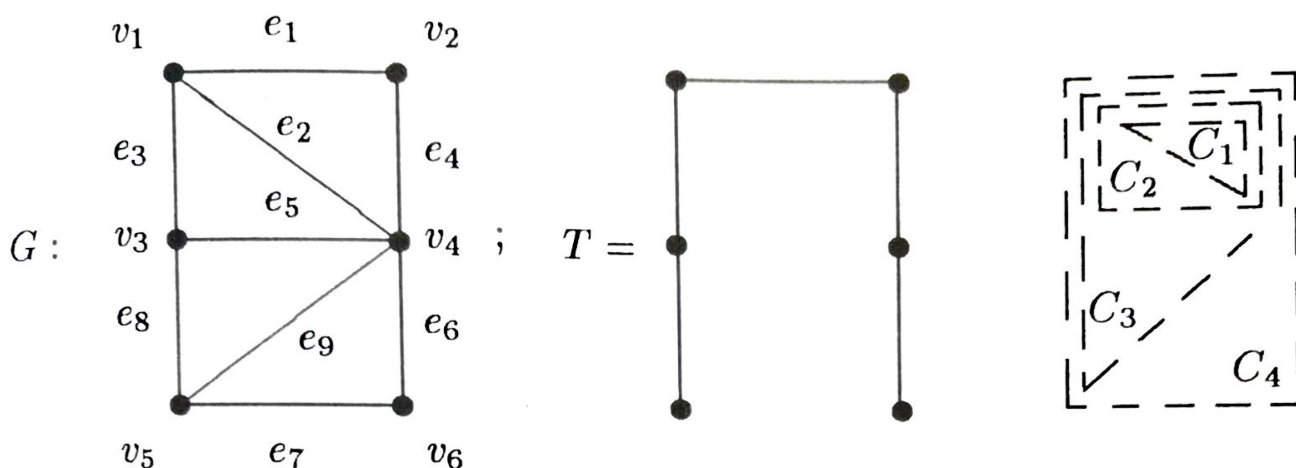
Ако множеството на тези цикли обозначим със $\mathcal{C}$, т.е.

$$\mathcal{C} = \{C_1,\ C_2,\ \ldots,\ C_{\mu(G)}\},$$

то всеки друг цикъл в графа G, непринадлежащ на $\mathcal{C}$, може да се изрази като линейна комбинация на циклите от $\mathcal{C}$. За целта е достатъчно всеки фундаментален цикъл C_i, $i = 1, \ldots, \mu(G)$ да се представи като m-мерен вектор, в който j-тата компонента е единица, когато j-тото ребро принадлежи на цикъла и нула в противен случай. Тогава, използвайки събирането по $mod\ 2$,

т.е. операцията $\oplus$, всеки цикъл може да се представи като сума по $mod\ 2$ от базови цикли.

Да разгледаме следния граф G и едно покриващо дърво T в G.



За $\mu(G)$ имаме $\mu(G) = m - n + k = 9 - 6 + 1 = 4$, т.е. $\mathcal{C} = \{C_1,\ C_2,\ C_3,\ C_4\}$ и те са показани на чертежа горе вдясно.

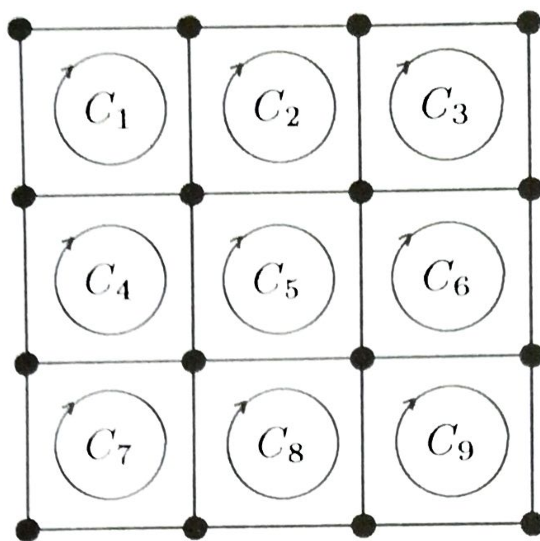
$$\begin{aligned} C_1 &= (110100000), \\ C_2 &= (101110000), \\ C_3 &= (101100011), \\ C_4 &= (101101110). \end{aligned}$$

Коментар:

1. Обръщаме внимание на факта, че броят на фундаменталните цикли е фиксиран, равен е на $\mu(G)$, но самите цикли не са определени еднозначно, а относно избраното покриващо дърво T. При друг избор на T се променя и множеството $\mathcal{C}$ на фундаменталните цикли.

2. В графа G могат да се намерят $\mu(G)$ на брой независими цикли, които не се получават по споменатия начин с добавяне на ребро към дървото T. Такива множества от независими цикли няма да считаме за множество от фундаментални цикли.

Например [2], показаното по-долу множество от $\mu(G) = 9$ независими цикли в графа G не може да се получи чрез добавяне на ребра към никое покриващо дърво T на този граф. Ето защо това множество не представлява множество от фундаментални цикли.



3. Наистина, както беше споменато, всеки цикъл може да се представи като линейна комбинация на цикли от $\mathcal{C}$. Обратното обаче не е вярно, т.е. не всяка сума по mod 2 от фундаментални цикли дава единствен прост цикъл. Например:

а) простият цикъл $\{e_2,\ e_3,\ e_5\}$ може да се представи като $C_1 \oplus C_2 = (011010000)$;

б) сумата $C_2 \oplus C_3 \oplus C_4 = (101111101)$ не съответства на прост цикъл, а на двата цикъла $\{e_1, e_3, e_4, e_5\}$ и $\{e_6, e_7, e_9\}$.

Оттук следва, че за да се породят всички прости цикли в графа, не е нужно да се взимат всичките $2^{\mu(G)} - 1$ комбинации от фундаментални цикли и да се събират по mod 2. Освен това, ако една сума от фундаментални цикли, да кажем $C_i \oplus C_j \oplus C_k \oplus \cdots$ не поражда прост цикъл, то не е ясно дали, ако към тази сума добавим сумата $C_p \oplus C_q \oplus C_r \oplus \ldots$ (тя също може да не поражда цикъл), няма да получим прост цикъл. Има предложени методи за отстраняване на комбинациите от фундаментални цикли, които не пораждат цикъл.

Понятията покриващо дърво и просто разрязващо множество (прост разрез) имат двойнствен характер, тъй като покриващото дърво T е минималното множество от ребра, свързващо всички върхове в графа G, а простият разрез е минималното множество от ребра, разделящо едни върхове от други. Именно поради тази двойнственост, всяко покриващо дърво в графа G имаше поне едно общо ребро с всеки прост разрез.

По аналогия с понятието фундаментален цикъл, отчитайки споменатата двойнственост, се въвежда и понятието *фундаментални разрези относно покриващото дърво* T — това са $n-1$ на брой прости разрези, всеки от които съдържа точно едно ребро, принадлежащо на дървото T. Не е друдно да се докаже

верността на следното твърдение:

	Ако T е покриващо дърво в неориентиран граф G, то фундаменталният разрез, определен от реброто $e_i \in T$ е образуван от e_i и онези ребра на G, непринадлежащи на T, които след добавяне към T дават фундаментален цикъл съдържащ e_i.

4. Силна свързаност

В края на този параграф ще разширим понятието свързаност. Да припомним, че ориентираният граф $G = (V, E)$ се нарича *свързан*, ако е свързан съответният му неориентиран дубликат. Подграфът на ориентирания граф G се нарича *компонента на* G, ако е компонента в съответния му неориентиран дубликат. На някои места в литературата тази "свързаност" се нарича *слаба свързаност*.

	Силна свързаност: Нека $G = (V, E)$ е ориентиран граф. 1. Два върха v_i и v_j се наричат *силно свързани върхове*, ако съществуват $(v_i - v_j)$ и $(v_j - v_i)$ пътища. 2. Графът G е *силно свързан*, ако са силно свързани всички негови върхове.

Очевидно релацията "силна свързаност" е релация на еквивалентност, т.е. е:

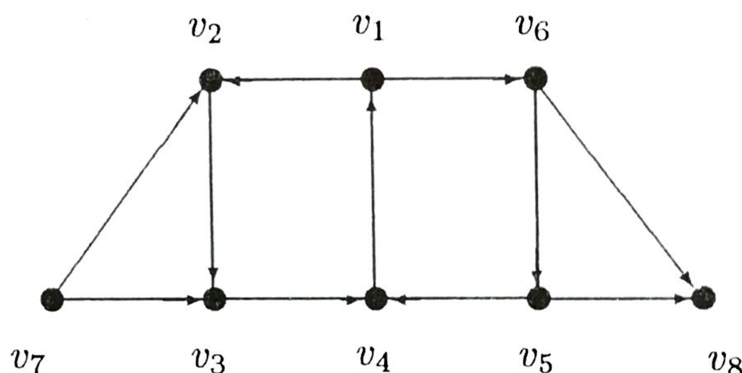
- рефлексивна — всеки връх е силно свързан със себе си;
- симетрична — ако v_i е силно свързан с v_j, то и v_j е силно свързан с v_i;
- транзитивна — ако v_i е силно свързан с v_s и v_s е силно свързан с v_j, то v_i е силно свързан с v_j (върховете v_i, v_s, v_j са силно свързани).

Всяка релация на еквивалентност, следователно и силната свързаност, разбива множеството където е дефинирана (в случая множеството от върхове) на непресичащи се класове на еквивалентност. Всеки клас ще се състои от силно свързани върхове, пораждащи силно свързан подграф на G. Именно тези

максимални силно свързани подграфи на G се наричат *силно свързани компоненти на* G.

Очевидно, ако G е силно свързан граф, той има една силно свързана компонента, а именно самият граф G.

Съществува една и само една силно свързана компонента, на която върхът v_i принадлежи. Ако допуснем, че принадлежи на две или повече компоненти, следва, че всеки връх от една компонента е взаимно достижим с всеки връх от друга компонента, т.е. обединението на тези компоненти би било силно свързан граф, което е невъзможно.



Черт. 1.10

Графът на черт. 1.10 не е силно свързан (макар да е свързан), тъй като например няма път между върховете v_1 и v_7 (път $v_7 - v_1$ има). Този граф има три силно свързани компоненти, породени от върховете $\{v_7\}$, $\{v_8\}$ и $\{v_1, v_2, v_3, v_4, v_5, v_6\}$.

Първите две компоненти са тривиални, а в третата цикличността дава възможност за достижимост (път) между всеки два върха.

Обърнете внимание, че в графа G може да има дъги, неучастващи в никоя силно свързана компонента. В графа G от черт. 1.10 такива са както дъгите, инцидентни с v_7, така и с v_8.

Понякога се използва и понятието *едностранно свързан граф* — това е граф, в който за всеки два върха v_i, v_j съществува поне един от пътищата $(v_i - v_j)$, $(v_j - v_i)$. Графът от черт. 1.10 е едностранно свързан. Аналогично, *едностранна компонента на* G — това е максималният едностранен подграф на G.

От дадените определения следва:

1. Едностранните компоненти могат да имат общи върхове за разлика от силно свързаните компоненти.

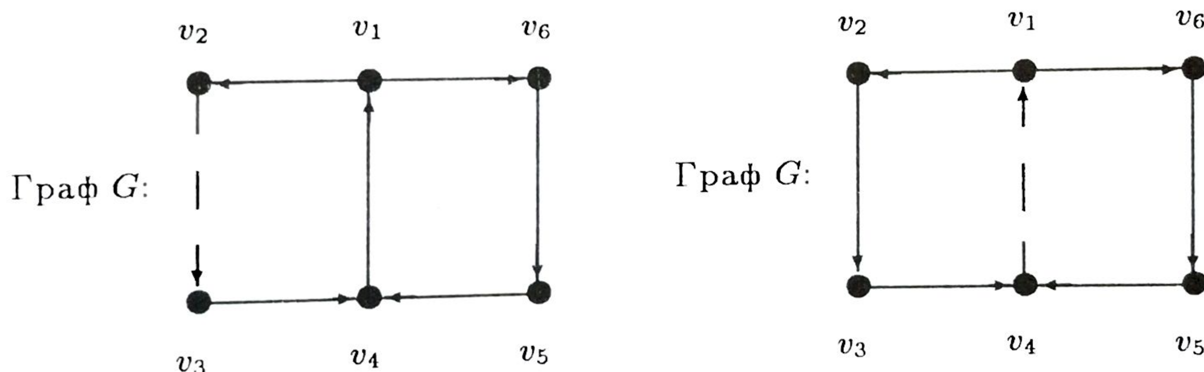
2. Всяка силно свързана компонента се съдържа в едностранна компонента.

3. Всяка едностранна компонента се съдържа в някоя (слабо) свързана компонента на G.

Илюстрирайте с примери горните три твърдения. В следващия параграф ще дадем алгоритъм за намиране на силно свързаните компоненти в един граф.

	Ориентираният граф G е *минимално свързан*, ако е силно свързан и след отстраняване на произволна дъга вече не притежава свойството силна свързаност.

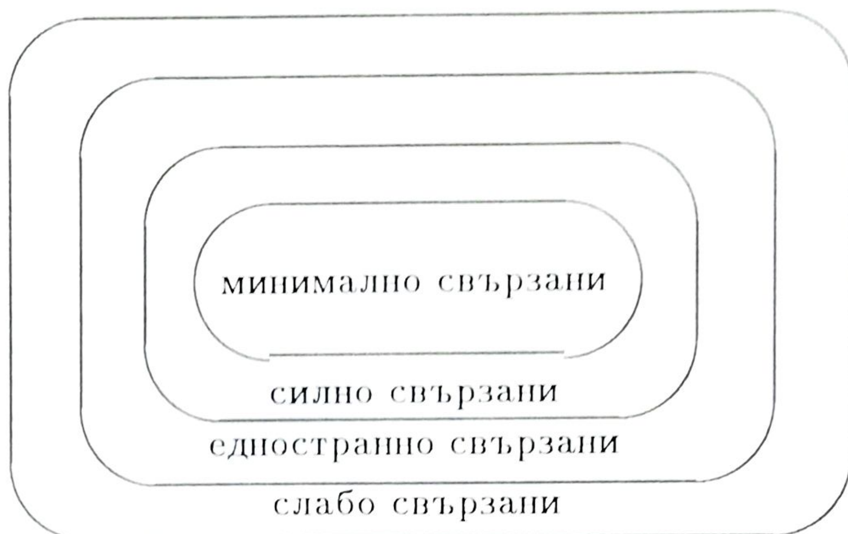
ПРИМЕР 3.2. Да разгледаме следния ориентиран граф (силно свързана компонента на графа от черт. 1.10):



Както вече споменахме, G е силно свързан граф. Нещо повече, този граф е минимално свързан, тъй като отстраняването на произволна дъга му отнема свойството силна свързаност. Например отстраняването на (v_2, v_3) води до подграф, в който няма $(v_2 - v_3)$ път. Казаното за (v_2, v_3) важи за всяка друга дъга на G.

По-особеното в случая е, че отстраняването на произволна дъга $(v_i, v_j) \neq (v_1, v_4)$ води до едностранно свързан граф (който не е силно свързан), а отстраняването на дъгата (v_1, v_4) не води до едностранно свързан, а до слабо свързан граф (вж. чертежа вдясно от пример 3.2. Няма $(v_2 - v_6)$ и $(v_6 - v_2)$ пътища.

За повече яснота по-долу са изобразени схематично класовете свързани графи.



От начина, по който дефинирахме минимално свързан граф е ясно, че такъв граф няма примки или паралелни дъги.

Ще дефинираме и *минимално свързан неориентиран граф* G като свързан граф, за който $G - e$ е несвързан (e — произволно ребро на G). Може да се докаже, че е вярно твърдението:

▷ *ТЕОРЕМА 3.16. Неориентираният граф G е минимално свързан тогава и само тогава, когато G е дърво.* ◁

От теорема 3.16 и 3.9 (всяко нетривиално дърво има поне два висящи върха) следва, че всеки минимално свързан неориентиран граф има поне два върха от степен 1 (висящи върхове). Ще докажем един аналог на теорема 3.9 за ориентирани графи.

▷ *ТЕОРЕМА 3.17. Ако G е минимално свързан ориентиран граф (нетривиален), той притежава поне два върха от степен 2.*

Доказателство: [3] Щом G е минимално свързан, по дефиниция той е силно свързан. Тогава степента на всеки връх ще бъде поне 2, тъй като за всеки връх има входяща и изходяща дъга. Очевидно за цикломатичното число (вж. края на параграф 2) на G ще имаме

$$\mu(G) \geq 1, \quad \mu(G) = m - n + k.$$

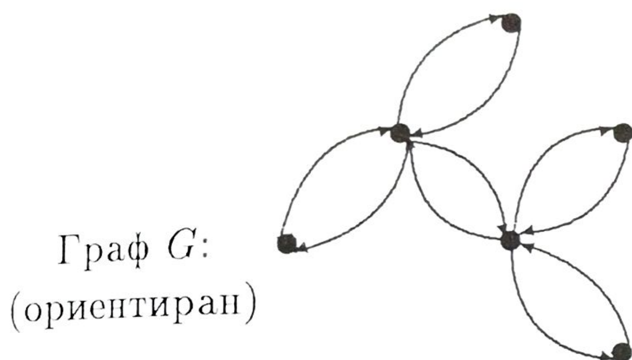
Доказателството на теоремата ще извършим с индукция по μ.

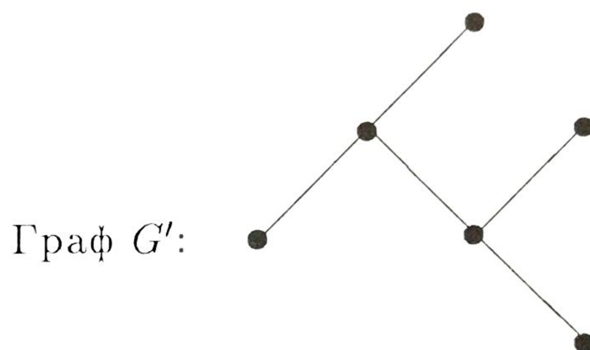
При $\mu = 1$ графът G се явява прост цикъл и в него очевидно ще съществуват поне 2 върха от степен 2.

Да допуснем, че теоремата е вярна за всеки минимално свързан ориентиран граф G, за който $\mu(G) \leq s - 1$, $s \geq 2$.

Ще докажем твърдението и за граф G, с цикломатично число $\mu = s$.

Първи случай: Всеки цикъл в G има дължина 2, както е показано на чертежа долу вляво.





В този случай всеки два върха са съединени с двойка противоположни дъги. Да разгледаме простия неориентиран граф G', който:

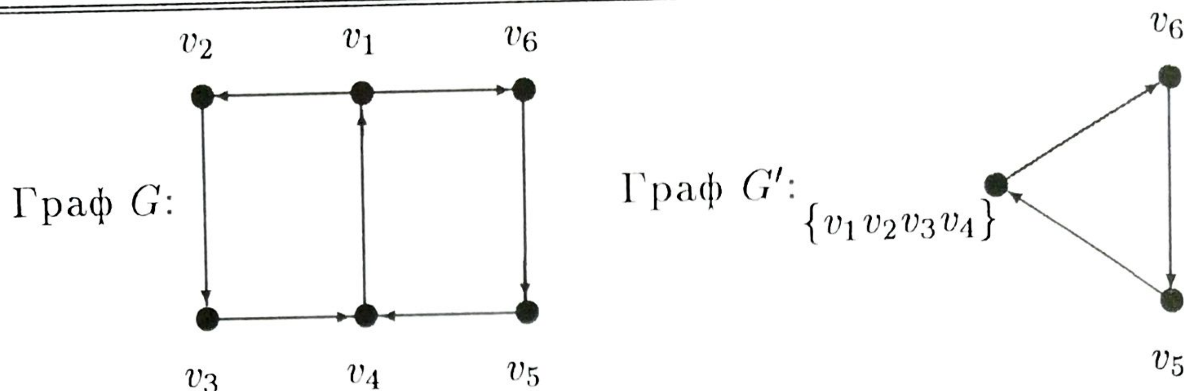
а) има същите върхове като G;

б) два върха в G' са съседни тогава и само тогава, когато тези върхове са съседни в G (вж. чертежа горе вдясно). Тъй като G е свързан и няма цикли с дължина повече от 2, очевидно G' ще бъде свързан граф без цикли, т.е. ще бъде дърво. Съгласно теорема 3.9 в G' ще има поне два върха от степен 1. Тези върхове в G ще имат степен 2, което доказва верността на теоремата в този случай.

Втори случай: В графа G има прост цикъл C с дължина $l > 2$.

Да разгледаме отново графа G от пример 3.2 и цикъла $C = (v_1, v_2, v_3, v_4, v_1)$, за който $l = 4$ (вж. черт. 1.11). Очевидно между всеки два съседни върха от цикъла има единствена дъга и между всеки два несъседни върха от цикъла няма дъги (обратното противоречи на минималната свързаност).

Да свием дъгите на цикъла C и означим с G' графа, който се получава (черт. 1.11).



Черт. 1.11

За ребрата m' и върховете n' на G' имаме
$m' = m - l$, m е броят дъги в G;
$n' = n - l + 1$, n е броят върхове на G;
откъдето следва

$$\mu(G') = (m - l) - (n - l + 1) + 1 = m - n.$$

Тъй като сме в случая $\mu(G) = s$, имаме

$$m - n + 1 = s \implies m - n = s - 1,$$

откъдето за $\mu(G')$ се получава

$$\mu(G') = s - 1.$$

Графът G' е минимално свързан и има цикломатично число $s - 1$. От индуктивното допускане следва, че в него поне два върха са от степен 2.

а) ако и двата върха са истински (не са получени след свиване на цикъла C), теоремата е доказана;

б) ако единият от върховете от степен 2 е свитият цикъл C, то в C ще има поне един врън от степен 2, което доказва и в този случай твърдението. ◁

ЗАДАЧА 3.2. Да се докаже, че ако T е произволно покриващо дърво за свързания граф G, то всяко висящо ребро на G се съдържа в T.

ЗАДАЧА 3.3. Ако T е дърво, то всеки невисящ връх на T се явява свързваща точка.

ЗАДАЧА 3.4. Да се докаже, че неразделимият граф G има цикломатично число 1 тогава и само тогава, когато той се явява цикъл.

ЗАДАЧА 3.5. Да се докаже, че необходимо и достатъчно условие графът G да бъде неразделим, е всеки две ребра да принадлежат на просто разрязващо множество.

ЗАДАЧА 3.6. Ако x и y са произволни ребра от цикъла C в графа G, може да се намери просто разрязващо множество S, за което $S \cap C = \{x,\ y\}$.

1.4. Матрично представяне на графи

Алгебрата се явява полезен и силен инструмент в теория на графите. В тази точка ще въведем някои основни матрици, свързани с графи и ще разгледаме някои свойства на тези матрици, разкриващи и изясняващи структурата на графа. Ще разгледаме основно матрици, свързани с ориентирани графи $G = (V, E)$, като голяма част от разсъжденията ще остават в сила и за неориентирани графи.

1. Матрица на инцидентност

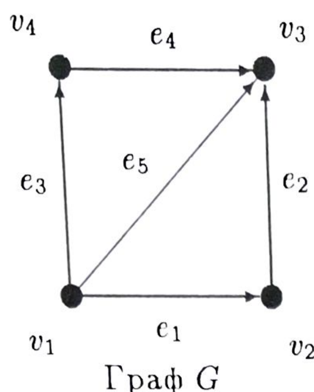
Матрица на инцидентност. Нека G е граф без примки с n върха и m дъги. Матрицата $A_I = (a_{ij})_{n,m}$, определена по следния начин:

$$a_{ij} = \begin{cases} 1, & \text{ако } v_i \text{ е начален връх за дъгата } e_j; \\ -1, & \text{ако } v_i \text{ е краен връх за дъгата } e_j; \\ 0, & \text{ако } v_i \text{ не е инцидентен с дъгата } e_j. \end{cases}$$

(При наличието на примка e_j за върха v_i може да се додефинира последното от горните три условия: $a_{ij} = 0$, ако v_i не е инцидентен с дъгата e_j или e_j се явява примка.)

Редовете на матрицата на инцидентност се наричат *вектори на инцидентност* за графа G.

ПРИМЕР 4.1.



Граф G

$$B = \begin{array}{c|c} & \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix} & \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix} \end{array}.$$

Матрица на инцидентност

Ако графът G е неориентиран, матрицата му на инцидентност може да се дефинира като

$$A_I = (a_{ij})_{n,m} = \begin{cases} 1, & \text{ако } v_i \text{ е инцидентен с реброто } e_j; \\ 0, & \text{в останалите случаи.} \end{cases}$$

С други думи, за неориентирани графи матрицата на инцидентност се дефинира аналогично както за ориентирани, с тази разлика, че всички елементи (-1) се заменят с $+1$.

Ясно е, че всеки стълб на матрицата на инцидентност съдържа точно два ненулеви елемента 1 и -1, тъй като всяка дъга е инцидентна с два различни върха — начало и край (ако в графа има примки, според забележката в дефиницията е възможно да има стълбове само с нулеви елементи). Следователно всеки ред на матрицата на инцидентност може да се определи чрез останалите $n-1$ реда, т.е. редовете на матрицата са линейно зависими и всеки $n-1$ от тях дават пълна информация за матрицата.

Всяка подматрица от $n-1$ реда A_C на матрицата на инцидентност се нарича *разрязана (пресечена) матрица на инцидентност*. Всяка такава подматрица е съответна на онзи връх, чийто ред е отрязан (отсъства) от матрицата A_I. Ясно е, че

$$rank(A_C) = rank(A_I) \leq n-1.$$

Лесно могат да се докажат следните шест твърдения.

▷ *ТЕОРЕМА 4.1. 1. Ако G е дърво, детерминантата на всяка разрязана матрица на инцидентност е ± 1.*
2. Ако G е свързан граф с n върха, рангът на неговата матрица на инцидентност е $n - 1$, т.е. рангът на графа $r(G) = rank(A_I)$.
3. Ако G е граф с n върха и k компоненти, то рангът на неговата матрица на инцидентност е $n - k$, т.е. рангът на графа $r(G) = rank(A_I)$.
4. Стълбовете на матрицата на инцидентност, съответстващи на дъги от прост цикъл, са линейно зависими.
5. Матрицата на инцидентност A_I на ориентирания граф е унимодулярна, т.е. детерминантата на всяка нейна квадратна подматрица е равна на 1, -1 или 0.
6. Нека G е свързан граф с n върха. Квадратната подматрица от ред $n-1$ на произволна пресечена матрица на инцидентност е неизродена тогава и само тогава, когато дъгите, съответстващи на стълбовете на подматрицата, образуват покриващо дърво. ◁

Верността на 1. лесно се установява с индукция по броя на върховете на дървото.

Тъй като всеки свързан граф G има поне едно покриващо дърво, от 1. следва, че за всяка пресечена матрица на инцидентност A_C съществува неизродена подматрица от ред $n - 1$, т.е. за свързания граф G, $rank(A_C) = n - 1$, и тъй като $rank(A_C) = rank(A_I)$, то следва верността на 2.

От верността на 2. директно следва верността на 3.

Верността на 5. лесно се установява с индукция по реда на квадратните подматрици на A_I.

Необходимостта в 6. се доказва с помощта на 4. и теорема 3.6. Достатъчността следва от 1.

Ще дадем едно приложение на матрицата на инцидентност за определяне броя на покриващите дървета на свързания граф G.

От линейната алгебра е известно, че ако P е матрица от ред $p \times q$, а Q е матрица от ред $q \times p$, $(p \leq q)$, то максималните квадратни подматрици на P и Q са от ред p. Нека максималната подматрица P_{max} на P се състои от стълбовете $i_1, i_2, ..., i_p$ на P, тогава максималната подматрица Q_{max} на Q, състояща се от редовете $i_1, i_2, ..., i_p$ на матрицата Q, се нарича *съответна*

подматрица на P_{max}. Например, ако

$$P = \begin{pmatrix} 5 & 1 & 2 \\ 3 & 4 & 1 \end{pmatrix} \quad \text{и} \quad Q = \begin{pmatrix} 7 & 8 \\ 0 & 1 \\ 3 & 4 \end{pmatrix}, \quad \text{то}$$

$$\text{на} \quad P_{max} = \begin{pmatrix} 5 & 2 \\ 3 & 1 \end{pmatrix}, \quad \text{съответната} \quad Q_{max} = \begin{pmatrix} 7 & 8 \\ 3 & 4 \end{pmatrix}.$$

Детерминантите на P_{max} и Q_{max} се наричат *главни детерминанти* съответно на P и Q. В сила е следната *теорема на Бине-Коши*.

▷ *ТЕОРЕМА 4.2. Ако* $P_{p\times q}$ *и* $Q_{q\times p}$, *то*

$\det(PQ) = \sum$ *(произведенията на съответните главни детерминанти,* $\det(P_{max})$ *и* $\det(Q_{max})$*).* ◁

Ако P и Q са горепосочените матрици, от теорема 4.2 имаме:

$$\det(PQ) = \begin{vmatrix} 5 & 2 \\ 3 & 1 \end{vmatrix} . \begin{vmatrix} 7 & 8 \\ 3 & 4 \end{vmatrix} + \begin{vmatrix} 5 & 1 \\ 3 & 4 \end{vmatrix} . \begin{vmatrix} 7 & 8 \\ 0 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 2 \\ 4 & 1 \end{vmatrix} . \begin{vmatrix} 0 & 1 \\ 3 & 4 \end{vmatrix} =$$

$$= -1.4 + 17.7 + (-7).(-3) = -4 + 119 + 21 = 136.$$

Като следствие от последната теорема, лесно се доказва следното твърдение:

▷ *ТЕОРЕМА 4.3. Нека* G *е свързан неориентиран граф и* A_C *е пресечена матрица на инцидентост за ориентирания граф* G^*, *получен чрез произволна ориентация на ребрата в* G. *Броят на покриващите дървета в графа* G *е* $\tau(G) = det(A_C.A_C^t)$.

Доказателство: От теорема 4.2 имаме

$(*)\quad \det(A_C A_C^t) = \sum$ *(произведенията на съответните главни детерминанти на* A_C *и* A_C^t*).*

Тъй като главните детерминанти на A_C и A_C^t имат еднакви стойности 1, -1 или 0 (вж.5 от теорема 4.1), то всяко събираемо, различно от 0 в горната сума има стойност 1. От друга страна главната детерминанта на A_C не е 0 тогава и само тогава, когато дъгите, съответстващи на стълбовете, образуват покриващо дърво. Оттук следва, че между ненулевите елементи в дясната част на $(*)$ и покриващите дървета на графа G съществува биективно изображение. С това теоремата е доказана. ◁

Да разгледаме графа от пример 4.1. Неговата матрица на инцидентност, пресечена по четвъртия ред, съответен на връх v_4, е:

$$A_C = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \left[\begin{array}{rrrrr} 1 & 0 & 1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & -1 \end{array}\right] \end{array}.$$

Следователно

$$A_C^t = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \text{и}$$

$$A_C.A_C^t = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 3 \end{bmatrix} \quad \Longrightarrow \quad \det(A_C.A_C^t) = 8.$$

Тогава броят на покриващите дървета (не говорим за ориентирани дървета) е 8: $\{e_1, e_2, e_3\}$; $\{e_1, e_2, e_4\}$; $\{e_1, e_3, e_5\}$; $\{e_1, e_4, e_5\}$; $\{e_1, e_3, e_4\}$; $\{e_2, e_3, e_4\}$; $\{e_2, e_3, e_5\}$; $\{e_2, e_4, e_5\}$. Тези покриващи дървета са съответни на подматриците на A_C от ред 3, които са различни от нула.

2. Матрица на съседство

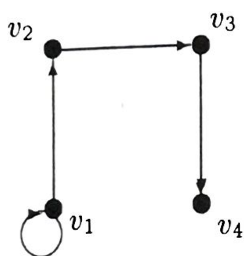
При задаване на графи е удобно и полезно да се използва и следната матрица:

Нека $G = (V, E)$ е произволен ориентиран граф с n върха без паралелни дъги. *Матрица на съседство* $B = (b_{ij})_{n,n}$ се нарича матрицата, чиито елементи се определят по следния начин:

$$b_{ij} = \begin{cases} 1, & \text{ако } (v_i, v_j) \in E; \\ 0, & \text{в останалите случаи.} \end{cases}$$

(В случай на неориентиран граф $b_{ij} = 1$ тогава и само тогава, когато съществува ребро между v_i и v_j).

ПРИМЕР 4.2.



Граф G

$$B = \begin{array}{c|cccc|} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 1 & 1 & 0 & 0 \\ v_2 & 0 & 0 & 1 & 0 \\ v_3 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 0 & 0 & 0 \end{array}.$$

Матрица на съседство

Матрицата на съседство определя напълно структурата на графа. Сумата от елементите в i-тия ред дава полустепента на изхода $d^+(v_i)$, а сумата от елементите в i-тия стълб дава полустепента на входа $d^-(v_i)$ на върха v_i.

Множеството от стълбове имащи 1 в реда v_i е множеството $\Gamma(v_i)$, а множеството редове имащи 1 в стълба v_i, е $\Gamma^{-1}(v_i)$.

Матрицата на съседство може да даде още информация за графа. Например, да разгледаме матрицата B^2, получена от умножаването на матрицата на съседство със себе си по правилото "ред по стълб", т.е. елементите на B^2 се получават по формулата

$$b_{ik}^2 = \sum_{j=1}^{n} b_{ij}.b_{jk}. \tag{4.1}$$

Едно събираемо в (4.1) е единица тогава и само тогава, когато $b_{ij} = b_{jk} = 1$, т.е. когато съществува път с дължина 2 от v_i до v_k през върха v_j. Следователно

$$b^2_{ik} = \text{"броя пътища с дължина 2 от } v_i \text{ до } v_k\text{"}.$$

Оттук по индукция следва, че елементите на матрицата $B^s = (b^s_{ik})_{n\times n}$ дават броя на пътищата от v_i до v_k с дължина s (не е задължително пътищата да са прости!).

За графа от пример 4.2 имаме:

$$B^2 = (b^2_{ik}) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad B^3 = (b^3_{ik}) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

За други нужди матрицата на съседство може да се определи и по малко по-различен начин, например ако всяка дъга в графа притежава тегло w_{ij}, вместо единици в матрицата на съседство се вписва теглото на съответната дъга w_{ij}. Матрицата на съседство се използва още при някои алгебрични методи за намиране на хамилтонови цикли.

3. Матрици на достижимост

Ще въведем две нови понятия — *матрица на достижимост* и *матрица на контрадостижимост* (reachability matrix и reaching matrix) [2]. Тези две матрици са важни, тъй като пряко кореспондират с моделирането на редица практически проблеми на езика на теория на графите. Например, ако имаме една система от хора (или други обекти), комуникиращи по определен начин помежду си (комуникацията може да бъде пренос на информация, съподчиненост и т.н.) на практика често възникват следните проблеми (задачи).

Какъв е най-големият брой хора (обекти), между които комуникацията е взаимна (например информацията е достижима между всеки два обекта)?

Често възниква и друг проблем. Да се намери някакво минимално, "базово" множество от хора, така че от тях информацията да бъде достижима за всички други обекти (хора). За да се опишат на езика на графите и успешно да се решават подобен род проблеми, е полезно въвеждането на понятията, които споменахме по-горе.

Матрица на достижимост. Това е матрицата $R = (r_{ij})_{n \times m}$, определена по следния начин:

$$r_{ij} = \begin{cases} 1, & \text{ако върха } v_j \text{ е достижим от } v_i; \\ 0, & \text{в противен случай.} \end{cases}$$

Да означим с $\mathcal{R}(v_i)$ множеството върхове в графа G, достижими от дадения връх v_i. Очевидно, ако ние намерим за всеки връх v_i съответното множество $\mathcal{R}(v_i)$, матрицата R лесно се построява — в нея r_{ij} е 1, когато $v_j \in \mathcal{R}(v_i)$ и 0 в противен случай. Лесно се съобразява, че

$$\mathcal{R}(v_i) = \{v_i\} \cup \Gamma(v_i) \cup \Gamma^2(v_i) \cup \ldots \cup \Gamma^s(v_i), \tag{4.2}$$

където $\Gamma(v_i)$ е множеството от върхове v_j, за които в графа съществува дъга (v_i, v_j), т.е. върховете, достижими от v_i чрез път с дължина 1;

Очевидно $\Gamma(\Gamma(v_i)) = \Gamma^2(v_i)$ ще бъде множеството от върхове, достижими от v_i чрез път с дължина 2 и т.н., $\Gamma^s(v_i)$ ще бъде множеството върхове, достижими от v_i с използването на път, чиято дължина е s.

От казаното следва, че по формула (4.2), изпълнявайки от ляво на дясно операцията обединение, ще намерим всички достижими от v_i върхове, като ще извършваме тази операция докато намереното (текущото) множество не престане да нараства по мощност, при поредното изпълнение на операцията обединение (от този момент нататък в множеството няма да се появяват нови елементи).

Очевидно е, че броят s на обединенията, които трябва да се изпълнят, е по-малък от броя n на върховете в графа G, както и че всеки диагонален елемент на R е равен на 1 (всеки връх е достижим от себе си чрез път с дължина 0).

Матрица на контрадостижимост. Това е матрицата $Q = (q_{ij})_{n \times n}$, определена по следния начин:

$$q_{ij} = \begin{cases} 1, & \text{ако от върха } v_j \text{ е достижим върха } v_i; \\ 0, & \text{в противен случай.} \end{cases}$$

Аналогично се определя множеството $Q(v_i)$ като множество върхове на графа G, такива че от всеки връх на това множество може да се достигне v_i. Ясно е, че по аналогия с формула (4.2),

$$Q(v_i) = \{v_i\} \cup \Gamma^{-1}(v_i) \cup \Gamma^{-2}(v_i) \cup \ldots \cup \Gamma^{-s}(v_i), \tag{4.3}$$

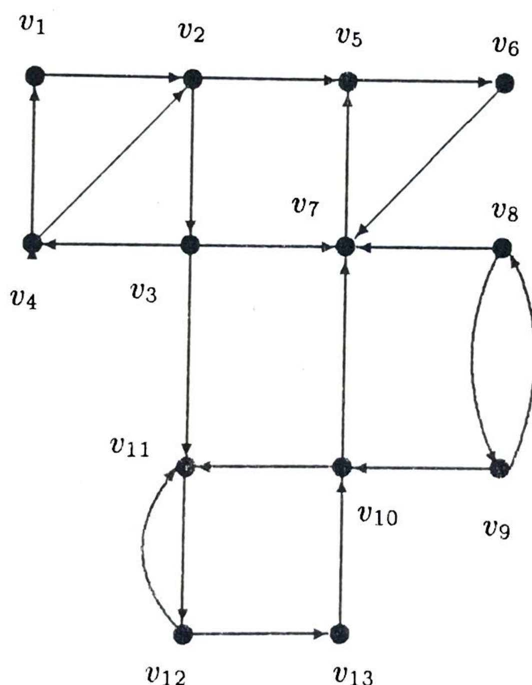
където $\Gamma^{-2}(v_i) = \Gamma^{-1}(\Gamma^{-1}(v_i))$ и т.н.

И в този случай операцията обединение се изпълнява от ляво на дясно докато не спре да се актуализира текущото множество $Q(v_i)$.

Очевидно в матрицата на контрадостижимост Q, елементът $q_{ij} = 1$, когато $v_j \in Q(v_i)$ и $q_{ij} = 0$ в противен случай.

Матрицата на контрадостижимост Q е транспонираната матрица на достижимост R, т.е. $Q = R^t$.

ПРИМЕР 4.3. За графа от черт. 1.12 да определим с формули (4.2) и (4.3) съответните матрици R и Q.



Черт. 1.12

Множествата на достижимост са:

$$\Gamma^0(v_1) = \{v_1\};$$
$$\Gamma^1(v_1) = \{v_2\};$$
$$\Gamma^2(v_1) = \{v_3, v_5\};$$
$$\Gamma^3(v_1) = \{v_4, v_7, v_{11}, v_6\};$$
$$\Gamma^4(v_1) = \{v_1, v_2, v_5, v_{12}, v_7\};$$
$$\Gamma^5(v_1) = \{v_2, v_3, v_5, v_6, v_{11}, v_{13}\};$$
$$\Gamma^6(v_1) = \{v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}, v_{10}\};$$
$$\Gamma^7(v_1) = \{v_4, v_7, v_{11}, v_6, v_1, v_2, v_5, v_{12}, v_{13}\}.$$

Постапната актуализация на множеството $\mathcal{R}(v_1)$ при поредното прилагане на операцията обединение (от ляво на дясно) от формула (4.2) дава:

$$\mathcal{R}(v_1) = \{v_1\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5, v_4, v_7, v_{11}, v_6\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}, v_{13}\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}, v_{13}, v_{10}\};$$
$$\mathcal{R}(v_1) = \{v_1, v_2, v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}, v_{13}, v_{10}\}.$$

$$\begin{aligned}\mathcal{R}(v_2) = \quad & \{v_2\} \cup \{v_3, v_5\} \cup \{v_4, v_7, v_{11}, v_6\} \cup \\ & \cup \{v_1, v_2, v_5, v_{12}, v_7\} \cup \{v_2, v_3, v_5, v_6, v_{11}, v_{13}\} \cup \\ & \cup \{v_3, v_5, v_4, v_7, v_{11}, v_6, v_{12}, v_{10}\} \cup \{v_4, v_7, v_{11}, v_6, v_1, v_2, v_5, v_{12}, v_{13}\} = \\ & = \{v_2, v_3, v_5, v_4, v_7, v_{11}, v_6, v_1, v_{12}, v_{13}, v_{10}\}.\end{aligned}$$

Аналогично получаваме:
$\mathcal{R}(v_3) = \mathcal{R}(v_4) = \mathcal{R}(v_1) = \mathcal{R}(v_2)$,
$\mathcal{R}(v_5) = \mathcal{R}(v_6) = \mathcal{R}(v_7) = \{v_5, v_6, v_7\}$,
$\mathcal{R}(v_8) = \mathcal{R}(v_9) = \{v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}\}$,
$\mathcal{R}(v_{10}) = \mathcal{R}(v_{11}) = \mathcal{R}(v_{12}) = \mathcal{R}(v_{13}) = \{v_5, v_6, v_7, v_{10}, v_{11}, v_{12}, v_{13}\}$.

Оттук лесно се определят (строят) матрицата на достижимост R и матрицата на контрадостижимост Q.

В следващата глава при разглеждането на въпроси, свързани с построяването на дървета, ще бъдат дадени други начини за намиране на множествата $\mathcal{R}(v_i)$ и $\mathcal{Q}(v_i)$, свързани с маркиране на върховете.

$R =$

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
v_1	1	1	1	1	1	1	1	0	0	1	1	1	1
v_2	1	1	1	1	1	1	1	0	0	1	1	1	1
v_3	1	1	1	1	1	1	1	0	0	1	1	1	1
v_4	1	1	1	1	1	1	1	0	0	1	1	1	1
v_5	0	0	0	0	1	1	1	0	0	0	0	0	0
v_6	0	0	0	0	1	1	1	0	0	0	0	0	0
v_7	0	0	0	0	1	1	1	0	0	0	0	0	0
v_8	0	0	0	0	1	1	1	1	1	1	1	1	1
v_9	0	0	0	0	1	1	1	1	1	1	1	1	1
v_{10}	0	0	0	0	1	1	1	0	0	1	1	1	1
v_{11}	0	0	0	0	1	1	1	0	0	1	1	1	1
v_{12}	0	0	0	0	1	1	1	0	0	1	1	1	1
v_{13}	0	0	0	0	1	1	1	0	0	1	1	1	1

Матрицата на контрадостижимост Q няма да разписваме, тъй като тя е транспонираната на R, т.е. $Q = R^t$.

Редица практически проблеми налагат да се строят така наречените матрици на *ограничена достижимост и контрадостижимост.* Те се намират точно по същия начин, с тази разлика, че при тях съществува ограничение за s, т.е. има някаква горна граница за дължината s на пътя, чрез който се реализира достижимостта или обратната достижимост (контрадостижимостта).

Не е трудно да се съобрази, че ако графът е транзитивен (т.е. от съществуването на дъгите (v_i, v_j) и (v_j, v_r) следва съществуването на дъгата (v_i, v_r)), неговата матрица на достижимост R съвпада с матрицата му на съседство B, ако в последната матрица по главния диагонал поставим единици.

Както анонсирахме в предишния параграф, сега ще дадем алгоритъм за намиране силно свързаните компоненти на графа G — максималните подграфи на G, в които всеки два върха са взаимно достижими.

АЛГОРИТЪМ ЗА НАМИРАНЕ НА СИЛНО СВЪРЗАНИТЕ КОМПОНЕНТИ.

Коментар:

1. Всеки връх v_i на графа G принадлежи само на една силно свързана компонента (вж. параграф 1.3).

2. Ако v_j е взаимно достижим с върха v_i, то очевидно съществува цикъл, съдържащ v_i и v_j.

3. Очевидно е вярно и обратното на 2., т.е. ако v_j е от цикъл, съдържащ v_i, то v_j е взаимно достижим с v_i (т.е. върховете са силно свързани).

От казаното следва, че един връх v_j е взаимно достижим (силно свързан) с върха v_i тогава и само тогава, когато върхът v_j е от цикъл, съдържащ v_i.

4. Множеството $\mathcal{R}(v_i) \cap \mathcal{Q}(v_j)$ се нарича *множество от съществени върхове относно върховете v_i и v_j* [2], [6]. Върховете от споменатото множество очевидно са такива, че от v_i до тях, както и от тях до v_j, има поне един път, т.е. върховете от $\mathcal{R}(v_i) \cap \mathcal{Q}(v_j)$ принадлежат поне на един път от v_i до v_j. Останалите върхове се наричат *несъществени*, тъй като тяхното отстраняване не влияе на $(v_i - v_j)$ пътя.

ИДЕЯ НА АЛГОРИТЪМА. От направения коментар е ясно: за да се намери единствената силна компонента на графа G, съдържаща върха v_i, трябва да се намерят всички върхове, учас-

тващи в поне един цикъл, съдържащ v_i (т.е. път с начало и край v_i); множеството $\mathcal{R}(v_i) \cap \mathcal{Q}(v_i)$ е множеството от всички върхове, до които има път от v_i и от които има път до v_i, т.е. всички върхове, участващи в поне един път от v_i до v_i, т.е. участващи в поне един цикъл, съдържащ v_i.

ОПИСАНИЕ НА АЛГОРИТЪМА. *СТЪПКА 1.* С помощта на (4.2) и (4.3) за произволен връх $v_i \in G = (V, \Gamma)$ се определят последователно $\mathcal{R}(v_i)$, $\mathcal{Q}(v_i)$ и $\mathcal{R}(v_i) \cap \mathcal{Q}(v_i)$. Последното множество еднозначно определя силната компонента на графа G, съдържаща върха v_i.

СТЪПКА 2. Отстраняват се всички върхове, участващи в намерената компонента и се разглежда подграфът G', породен от множеството върхове $\langle V - R(v_i) \cap Q(v_i) \rangle$. Полагаме $G := G'$ и се връщаме на *стъпка 1*, докато всички върхове на изходния граф G не се групират в силно свързани компоненти. По този начин графът G се *разбива* на силно свързаните си компоненти [7].

Изложената процедура може да се осъществи лесно, ако се използват матриците R и Q [2]. Да означим с $R \otimes Q$ матрицата, която се получава след поелементното умножаване на матриците R и Q, т.е.

$$R \otimes Q = (r_{ij} . q_{ij})_{n \times n}.$$

Очевидно редът, съответстващ на върха v_i в матрицата $R \otimes Q$, ще съдържа единици в тези стълбове v_j, за които v_i и v_j са взаимно достижими и ще съдържа нули на останалите места, т.е. два върха ще се намират в една и съща силно свързана компонента тогава и само тогава, когато съответните им редове (или стълбове) в матрицата $R \otimes Q$ са еднакви. Върховете, чиито редове имат единица в стълба v_j, ще образуват множеството върхове на силно свързаната компонента, съдържаща v_j. Оттук следва, че матрицата $R \otimes Q$ може да се преобразува чрез разместване на редове и стълбове в блочно-диагонална матрица, в която всяка диагонална подматрица съответства на силно свързана компонента и се състои само от единици (останалите елементи на блочно-диагоналната матрица са нули).

За графа от пример 4.3 матрицата $R \otimes Q$ след съответните

преобразувания има вида:

$$R \otimes Q = \begin{array}{c|ccccccccccccc|} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} \\ \hline v_1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_2 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_3 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_4 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_5 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_6 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_7 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ v_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ v_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ v_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ v_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ v_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array},$$

т.е. съответните силно свързани компоненти на графа са:

$$\{v_1, v_2, v_3, v_4\}; \quad \{v_5, v_6, v_7\}; \quad \{v_8, v_9\}; \quad \{v_{10}, v_{11}, v_{12}, v_{13}\}.$$

Да разгледаме сега въпроса за намирането на минимално множество върхове в един граф $G = (V, E)$, от които са достижими всички върхове на графа, т.е. намирането на множество от върхове B, за което:

$$(4.4) \qquad \mathcal{R}(B) = \bigcup_{v_i \in B} \mathcal{R}(v_i) = V \quad \text{и} \quad \forall B' \subset B \ (\mathcal{R}(B') \neq V).$$

Ако в графа $G = (V, E)$ има покриващо ориентирано дърво с корен v_i (v_i се нарича още *корен на графа* G), този корен v_i ще се явява търсеното множество B. Множеството B, определено с (4.4), се нарича *база*.

Ще дадем още една дефиниция за база, еквивалентна на (4.4).

▷ *ТЕОРЕМА 4.4. Множеството от върхове B на графа $G = (V, E)$ е база тогава и само тогава, когато $\mathcal{R}(B) = V$ и в B няма връх, достижим от друг връх на B.*

Доказателство: *Необходимост:* Нека B е база в G, т.е. в сила е (4.4). Да допуснем, че съществува връх $v_j \in B$, който

е достижим от върха $v_i \in B$, т.е. $v_j \in \mathcal{R}(v_i)$ и $v_i \neq v_j \in B$. Тогава очевидно върховете достижими от v_j, ще бъдат достижими и от v_i, т.е. $\mathcal{R}(v_j) \subseteq \mathcal{R}(v_i)$. Оттук следва, че

$$\mathcal{R}(B - v_j) = \bigcup_{v_i \in B - v_j} \mathcal{R}(v_i) = V,$$

което противоречи на второто условие в (4.4).

Достатъчност: Нека сега за множеството от върхове B имаме $R(B) = V$ и в B няма връх, достижим от друг връх на B, т.е. за всеки два различни върха от B имаме $v_j \notin \mathcal{R}(v_i)$.

Ще докажем, че е налице (4.4), за което е достатъчно да покажем само второто условие на (4.4).

Да допуснем противното. Без ограничение на общността да допуснем, че истинското подмножество $B' = B - v_j$, $v_j \in B$ притежава свойството $\mathcal{R}(B') = V$. Тогава $\bigcup_{v_i \in B - v_j} \mathcal{R}(v_i) = V$. Върхът $v_j \in V$, следователно $v_j \in \bigcup_{v_i \in B - v_j} \mathcal{R}(v_i)$, откъдето $v_j \in \mathcal{R}(v_i)$, $v_i \neq v_j$, което противоречи на направеното предположение в условието. С това теоремата е доказана. ◁

СЛЕДСТВИЕ 1. Никои два върха от базата B не принадлежат на една и съща силно свързана компонента на графа G.

СЛЕДСТВИЕ 2. Нека G е произволен граф и B е негова произволна база. Всички върхове v_j на графа G, чиято полустепен на входа е нула, т.е. $d^-(v_j) = 0$, принадлежат на базата B.

Верността на следствие 2 е очевидна.

Обратното твърдение на следствие 2 не е вярно, както ще се убедим по-късно.

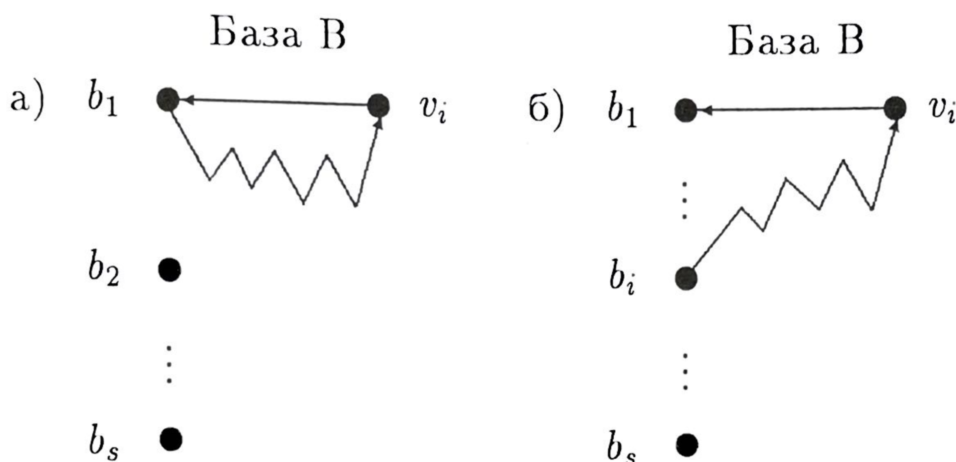
▷ *ТЕОРЕМА 4.5. Нека G е ацикличен граф. В G съществува единствена база B, съдържаща всички върхове на графа с полустепен на входа нула.*

Доказателство: От следствие 2 е ясно, че всички върхове на G с нулева полустепен на входа са от B. За да обосновем верността на теоремата, е достатъчно да докажем, че в базата B на ацикличния граф G няма върхове v_j, за който $d^-(v_j) \neq 0$.

Да допуснем, че $B = \{b_1, b_2, ..., b_s\}$ е база, в която има поне един връх, например b_1, за който $d^-(b_1) \neq 0$. Следователно съществува дъга (v_i, b_1). Но щом B е база, върхът v_i е достижим от някое b_i, $i \in \{1,\ 2,\ ...,\ s\}$.

а) Нека $b_i = b_1$. Тогава съществува цикъл, съдържащ b_1, което е невъзможно поради ацикличността на G (вж. черт. 1.13 а));

б) Нека $b_i \neq b_1$. Тогава има път от b_i до b_1, което противоречи на теорема 4.4 (вж. черт. 1.13 б)).

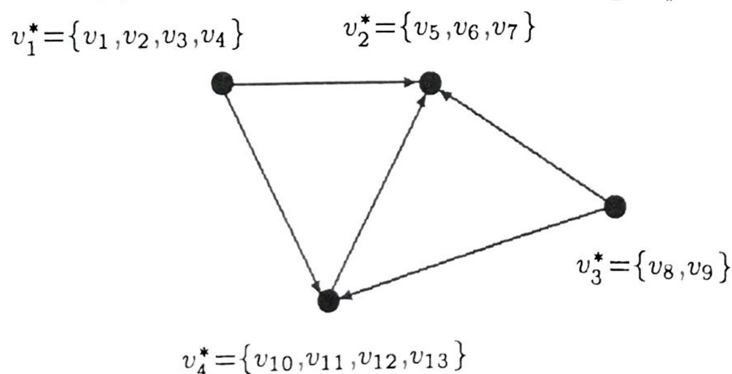


Черт. 1.13

И така всички върхове на ацикличния граф G с нулева полустепен на входа, и само те, са от базата B, което доказва единствеността на базата B. ◁

Ще въведем още едно важно и полезно понятие в теория на графите. Нека $G = (V, \Gamma)$. Да разгледаме графа $G^* = (V^*, \Gamma^*)$, определен по следния начин: всеки негов връх представлява силно свързана компонента на G (на различните компоненти на G съответстват различни върхове на G^*); дъгата (v_i^*, v_j^*) съществува в G^* тогава и само тогава, когато в G съществува дъгата (v_i, v_j), за която v_i принадлежи на компонентата, съответна на v_i^*, а v_j — на компонентата, съответна на v_j^*. Графът G^* се нарича *кондензация на графа* G [2].

На черт. 1.14 е дадена кондензацията на графа от черт. 1.12.



Черт. 1.14

Очевидно е, че кондензацията G^* на G е ацикличен граф. Допускането, че в G^* има цикъл означава, че всички върхове от цикъла са взаимно достижими, откъдето следва, че върховете на цикъла са в някоя силно свързана компонента на G^*, а оттам и на G, което противоречи на определението за кондензация.

Следствията на теорема 4.4 и теорема 4.5 дават възможност да формулираме следния

АЛГОРИТЪМ ЗА НАМИРАНЕ НА БАЗА В ГРАФА G.

СТЪПКА 1. Намират се силно свързаните компоненти на G с помощта на дадения вече алгоритъм.

СТЪПКА 2. Намира се кондензацията на графа G.

СТЪПКА 3. Намира се единствената база B^* на ацикличния граф G^*, която се състои от всички върхове v_i^* с нулева полустепен на входа.

СТЪПКА 4. От всяка силна компонента на графа G, съответстваща на връх от базата B^* се взима точно един произволен връх. Край.

За графа от черт. 1.12 съответната кондензация е на черт. 1.14. Единствената база за ацикличния граф от черт. 1.14 е базата $B = \{v_1^*, v_3^*\}$. Следователно за изходния граф G са възможни следните бази: $\{v_1, v_8\}$, $\{v_1, v_9\}$, $\{v_2, v_8\}$, $\{v_2, v_9\}$, $\{v_3, v_8\}$, $\{v_3, v_9\}$, $\{v_4, v_8\}$, $\{v_4, v_9\}$.

От казаното е очевидно:

> Всеки две бази на произволния граф G имат един и същ брой върхове, равен на броя на върховете с нулева полустепен на входа от кондензацията (т.е. броят на върхове от базата B^* на G^*).

4. Матрица (вектор) на степените

Да разгледаме следната матрица (вектор):

$$D = (d(v_i))_{1\times n} = (d(v_1),\ d(v_2),\ \ldots,\ d(v_n)),$$

чийто елементи са неотрицателни цели числа. Всяка такава матрица (вектор) се нарича *матрица на степените*, ако същес-

твува граф с върхове v_1, v_2, ..., v_n, чийто степени са съответно $d(v_1)$, $d(v_2)$, ..., $d(v_n)$.

Ще опишем и обосновем алгоритъм за намиране на неориентиран прост граф G при зададена матрица на степените. Алгоритъмът ще намира прост граф G, ако той съществува или ще установява несъществуването на граф, ако последователността $d(v_1)$, $d(v_2)$, ..., $d(v_n)$ не може да бъде последователност от степени на върхове.

ИДЕЯ НА АЛГОРИТЪМА. Разглежда се последователността $D = (d(v_1),\ d(v_2),\ ...,\ d(v_n))$, като $d(v_1) \geq d(v_2) \geq ... \geq d(v_n)$, което лесно се постига с пренареждане на върховете, ако това се налага. Избира се произволно $d(v_k) \neq 0$. "Отнема (занулява) се" степента $d(v_k)$, като върхът v_k се свързва чрез ребро с първите $d(v_k)$ върха от последователността, различни от v_k.

От степените на върховете, свързани с v_k, се отнема 1. По този начин от D се получава една нова последователност D_1 от числа, наречена *остатъчна последователност.* Чрез пренареждане на върховете (ако това се налага) в остатъчната последователност числата отново са в ненарастващ ред. Тази процедура се повтаря, докато:

а) се получи остатъчна последователност $D_i = (0,\ 0,\ ...,\ 0)$, което означава, че има прост граф с матрица (вектор) на степените D или

б) се получи остатъчна последователност D_i с отрицателна компонента, което означава, че D не може да бъде матрица на степените, т.е. не съществува прост граф G с такива степени на върховете.

ПРИМЕР 4.4. Да разгледаме последователността

$$D = \left(\frac{v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6}{4, \quad 2, \quad 2, \quad 2, \quad 3, \quad 3}\right).$$

Пренареждаме върховете, така че последователността да стане ненарастваща.

$$D = \left(\frac{v_1 \quad v_5 \quad v_6 \quad v_2 \quad v_3 \quad v_4}{4, \quad 3, \quad 3, \quad 2, \quad 2, \quad \boxed{2}}\right).$$

Отнемаме (зануляваме) степента на върха v_4, като го свързваме с ребро с първите $d(v_4)$, т.е. с първите 2 върха v_1 и v_5. Получава се следната остатъчна последователност:

$$D_1 = \left(\frac{v_1 \quad v_5 \quad v_6 \quad v_2 \quad v_3 \quad v_4}{3, \quad 2, \quad 3, \quad 2, \quad 2, \quad 0}\right).$$

Пренареждат се върховете в D_1 до вида

$$D_1 = \left(\frac{v_1 \quad v_6 \quad v_5 \quad v_2 \quad v_3 \quad v_4}{3, \quad \boxed{3}, \quad 2, \quad 2, \quad 2, \quad 0}\right).$$

Отнема се (занулява се) степента на произволен връх, например v_6, като v_6 се свързва с първите $d(v_6)$, т.е. с първите 3 върха без да се брои върхът v_6, т.е. v_6 се свързва с v_1, v_5 и v_2. По този начин се получава следната остатъчна последователност:

$$D_2 = \left(\frac{v_1 \quad v_6 \quad v_5 \quad v_2 \quad v_3 \quad v_4}{2, \quad 0, \quad 1, \quad 1, \quad 2, \quad 0}\right), \quad \text{т.е.} \quad D_2 = \left(\frac{v_1 \quad v_3 \quad v_5 \quad v_2 \quad v_4 \quad v_6}{2, \quad \boxed{2}, \quad 1, \quad 1, \quad 0, \quad 0}\right).$$

Отнема се степента на v_3 и се получава:

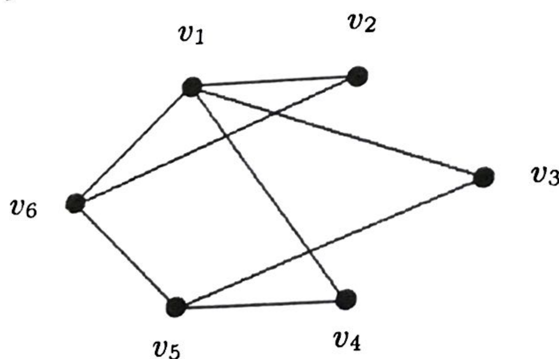
$$D_3 = \left(\frac{v_1 \quad v_3 \quad v_5 \quad v_2 \quad v_4 \quad v_6}{1, \quad 0, \quad 0, \quad 1, \quad 0, \quad 0}\right), \quad \text{т.е.} \quad D_3 = \left(\frac{v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6}{1, \quad \boxed{1}, \quad 0, \quad 0, \quad 0, \quad 0}\right).$$

Отнема се степента на v_2 и се получава:

$$D_4 = \left(\frac{v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6}{0, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0}\right).$$

Следователно съществува прост граф G с матрица (вектор) на степените $D = (4,\ 3,\ 3,\ 2,\ 2,\ 2)$. Графът G се получава като резултат от изпълнението на отделните стъпки, съответстващи на отнемането (зануляването) на степените:

1. Върхът v_4 се свързва с v_1 и v_5;
2. Върхът v_6 се свързва с v_1, v_5 и v_2;
3. Върхът v_3 се свързва с v_1 и v_5;
4. Върхът v_2 се свързва с v_1.



Въпрос: Единствен ли е полученият прост граф G? (защо?).

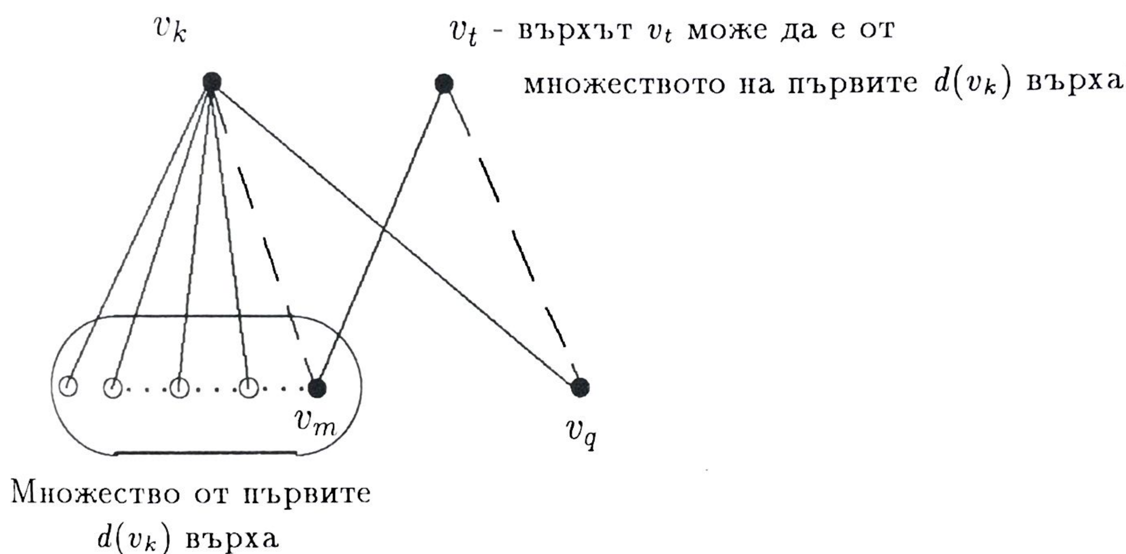
Обосновката на предложения алгоритъм се базира на следния резултат, получен от Wang и Kleitman [8].

▷ *ТЕОРЕМА 4.6. Ако последователността* $D = (d(v_1),\ d(v_2),\ \ldots,\ d(v_n))$, $d(v_1) \geq d(v_2) \geq \ldots \geq d(v_n)$, *се явява матрица на степените на прост граф, то и остатъчната последователност, получена след отнемането на степента* $d(v_k)$ *притежава същото свойство.*

Доказателство: За целта е достатъчно да покажем, че съществува прост граф с матрица на степените $D = (d(v_1), d(v_2), ..., d(v_n))$, в който върха v_k е съседен с първите $d(v_k)$ върха, различни от v_k. Да допуснем противното.

Избираме измежду графите с горната матрица на степените онзи прост граф G, в който върхът v_k е съседен с максималния брой върхове измежду първите $d(v_k)$ върха и не е съседен с поне един от тях — например v_m (виж чертежа по-долу). Щом v_k не е съседен с v_m, v_k ще бъде съседен с връх v_q, който не е измежду първите $d(v_k)$ върха. Тогава

$$d(v_m) \geq d(v_q) \geq 1.$$



Множество от първите $d(v_k)$ върха

Първи случай:
Нека $d(v_m) = 1 \implies d(v_q) = 1$. Тогава съществува връх v_t, съседен на v_m. Върхът v_t е различен от v_k по допускане, $v_t \neq v_q$ поради $d(v_q) = 1$ и $v_t \neq v_m$ поради липсата на примки в графа. Освен това върхът v_t (съседен на v_m) не е съседен на v_q поради $d(v_q) = 1$.

Втори случай: Нека $d(v_m) \geq 2$, т.е. съществуват поне 2 съседни на v_m върха. Следователно съществува поне един връх v_t ($\neq v_q$ и v_m), който е съседен на v_m. Ако допуснем, че всички такива върхове v_t са съседни на v_q, за степента на върха v_q ще получим

$$d(v_q) = 1 + d(v_m),$$

от където поради $d(v_m) \geq d(v_q)$ стигаме до противоречието

$$d(v_m) \geq d(v_m) + 1.$$

И така, във всички случаи съществува връх v_t ($\neq v_q$ и v_m), съседен на v_m и несъседен на v_q. Ако заменим ребрата (v_m, v_t) и (v_k, v_q) съответно с ребрата (v_k, v_m) и (v_q, v_t), ще получим граф G', в който множеството на първите $d(v_k)$ върха (различни от v_k), съседни с v_k, съдържа един елемент повече, което противоречи на задаването на графа G (виж началото на доказателството). ◁

В [9] са дадени необходими и достатъчни условия една последователност от числа да бъде степени на върховете на граф, но тези условия са от неалгоритмичен тип.

Ако в гореописания алгоритъм на всяка стъпка се отнема (занулява) най-малката различна от нула остатъчна степен, то с помощта на индукция може да се покаже, че получаваният граф е свързан, ако

$$\forall i, \quad d(v_i) \geq 1 \quad u \quad \sum_{i=1}^{n} d(v_i) \geq 2(n-1).$$

5. Матрици на циклите и разрезите

В тази подточка на параграфа ще използваме думите разрез и цикъл, като ще визираме прост разрез и прост цикъл (контур). Нека G е граф, имащ m ребра (дъги).

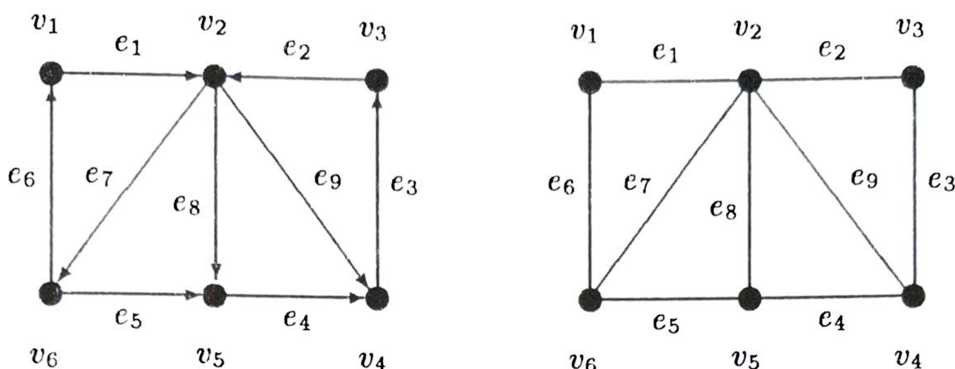
Матрицата $C = (c_{ij})$, имаща m стълба и толкова редове, колкото са циклите в графа G, чийто елементи се определят по следния начин:

$$\underset{\text{(ориентиран граф)}}{c_{ij}} = \begin{cases} 1, & \text{ако } j\text{-тата дъга влиза в } i\text{-тия цикъл и ориентацията на дъгата съответства на ориентацията на цикъла;} \\ -1 & \text{ако } j\text{-тата дъга влиза в } i\text{-тия цикъл и ориентацията на дъгата не съответства на ориентацията на цикъла;} \\ 0, & \text{в противен случай,} \end{cases}$$

$$\underset{\text{(неориентиран граф)}}{c_{ij}} = \begin{cases} 1, & \text{ако } j\text{-тото ребро влиза в } i\text{-тия цикъл;} \\ 0, & \text{в противен случай,} \end{cases}$$

се нарича *цикломатична матрица (матрица на циклите).*

ПРИМЕР 4.5. Да разгледаме следния ориентиран граф G и неговия неориентиран дубликат.



Ако за всички цикли приемем "единна" ориентация по посока на часовниковата стрелка, тогава за трите цикъла $\{e_1, e_8, e_5, e_6\}$, $\{e_2, e_3, e_9\}$ и $\{e_2, e_3, e_4, e_8\}$ подматрицата на C е следната:

$$\begin{array}{l} \\ \text{Цикъл 1} \\ \text{Цикъл 2} \\ \text{Цикъл 3} \end{array} \begin{array}{c} \begin{array}{ccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \end{array} \\ \left[\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 \end{array}\right] \end{array}.$$

Съответната подматрица за неориентирания граф се получава, ако в горната подматрица "-1" се смени с "$+1$".

Ще дефинираме сега една важна подматрица на цикломатичната матрица, базирайки се на въведеното в предишната подточка понятие — фундаментален цикъл.

Матрица на фундаменталните цикли в неориентиран граф G относно покриващото дърво T се нарича матрицата $C_f = (c_{ij})$, имаща $\mu(G)$ реда и m стълба, в която $c_{ij} = 1$, ако реброто e_j принадлежи на фундаменталния цикъл C_i и $c_{ij} = 0$, в противен случай.

Лесно се съобразява, че ако ребрата, непринадлежащи на дървото T, номерираме последователно от 1 до $\mu(G)$, а ребрата на дървото T номерираме от $\mu(G) + 1$ до m, то матрицата на фундаменталните цикли C_f има вида:

$$C_f = (I|C_{12}), \tag{4.5}$$

където I е единична матрица. Това е така, защото всеки фундаментален цикъл C_i съдържа точно едно ребро, непринадлежащо на T и циклите можем да номерираме съответно чрез номера на реброто, непринадлежащо на дървото. Оттук следва, че всички

единици в първата $(\mu(G) \times \mu(G))$ подматрица лежат на главния диагонал. (При ориентирани графи ориентацията на фундаменталния цикъл съвпада с ориентацията на определящата го дъга, непринадлежаща на дървото T.)

За неориентирания граф от пример 4.5 и покриващото дърво $T = \{e_1, e_2, e_4, e_5, e_6\}$, фундаменталните цикли и матрицата на фундаменталните цикли съответно са:

$$
\begin{aligned}
C_1 &= (1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0) && \text{— съответен на реброто } e_3 \notin T;\\
C_2 &= (1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0) && \text{— съответен на реброто } e_7 \notin T;\\
C_3 &= (1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0) && \text{— съответен на реброто } e_8 \notin T;\\
C_4 &= (1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1) && \text{— съответен на реброто } e_9 \notin T;
\end{aligned}
$$

(4.6)

$$
C_f = \begin{array}{c} \\ \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array}
\begin{array}{cccc|ccccc}
e_1' & e_2' & e_3' & e_4' & e_5' & e_6' & e_7' & e_8' & e_9 \\
e_3 & e_7 & e_8 & e_9 & e_1 & e_2 & e_4 & e_5 & e_6 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1
\end{array}.
$$

От (4.5) следва, че рангът на матрицата C_f е равен на $m - n + 1$ (щом има покриващо дърво T, графът G е свързан и цикломатичното му число $\mu(G) = m - n + 1$). Тъй като C_f е подматрица на матрицата на циклите C, то

$$rank(C) \geq rank(C_f) = m - n + 1.$$

Може да се докаже верността на следното твърдение:

▷ *ТЕОРЕМА 4.7. Рангът на цикломатичната матрица C съвпада с цикломатичното число на графа G, т.е. $rank(C) = \mu(G) = m - n + k$ (при свързан граф $rank(C) = \mu(G) = m - n + 1$, т.е. $rank(C) = rank(C_f)$).* ◁

Матрица на разрезите: В графа G с m ребра (дъги) матрицата $S = (s_{ij})$, имаща m стълба и толкова реда, колкото са на брой разрезите в графа, чийто елементи s_{ij} се определят по следния

начин:

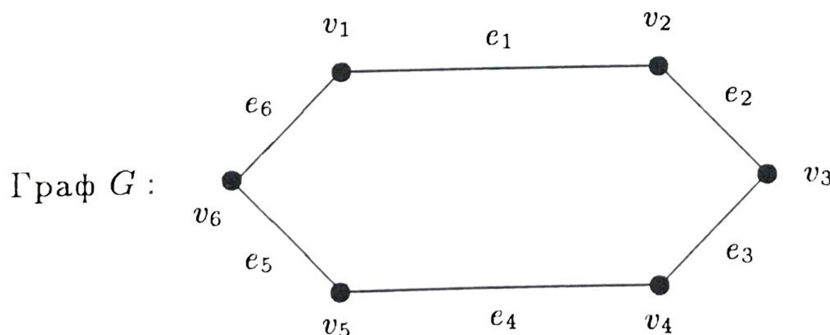
$$s_{ij} \text{ (ориентиран граф)} = \begin{cases} 1, & \text{ако } j\text{-тата дъга влиза в } i\text{-тия разрез и ориентацията на дъгата съвпада с тази на разреза;} \\ -1 & \text{ако } j\text{-тата дъга влиза в } i\text{-тия разрез и ориентацията на дъгата не съвпада с ориентацията на разреза;} \\ 0, & \text{в противен случай,} \end{cases}$$

$$s_{ij} \text{ (неориентиран граф)} = \begin{cases} 1, & \text{ако } j\text{-тото ребро влиза в } i\text{-тия разрез;} \\ 0, & \text{в противен случай,} \end{cases}$$

се нарича *матрица на разрезите.*

Ясно е, че ако знаем матрицата на разрезите в ориентирания граф G, съответната матрица за неговия неориентиран дубликат се получава, като заменим " -1" с " $+1$".

ПРИМЕР 4.6. Да разгледаме следния неориентиран граф G:



Очевидно всички прости разрязващи множества и съответните разрези са двуелементни, тъй като отстраняването на едно ребро не нарушава свързаността на графа, отстраняването на всеки две ребра нарушава свързаността, а всеки три ребра нарушават свързаността, но не са минимално множество с това свойство. Матрицата на разрезите S има вида

$$S = \begin{array}{r} \\ \text{Разрез 1} \\ \text{Разрез 2} \\ \text{Разрез 3} \\ \dots \\ \text{Разрез 5} \\ \text{Разрез 6} \\ \dots \\ \text{Разрез 15} \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array} .$$

В матрицата на инцидентност ненулевите елементи в един ред, съответен на върха v определят инцидентните с върха v дъги, а тези дъги образуват разрез $\langle v, V - v\rangle$. Тогава от дадените дефиниции за матрица на инцидентност и матрица на разрезите следва, че редът в матрицата S, съответстващ на разреза $\langle v, V - v\rangle$ (с ориентация от v към $V - v$) съвпада с реда на матрицата на инцидентност A_I, съответен на върха v. Оттук следва, че матрицата на инцидентност A_I е подматрица на матрицата на разрезите S. Ще докажем, че $rank(S) = rank(A_I)$.

▷ ТЕОРЕМА 4.8. *Всеки ред в матрицата на разрезите S може да се изрази по два начина като линейна комбинация от редовете на матрицата A_I, като и в двата случая ненулевите коефициенти в линейната комбинация са "$+1$" или "-1".*

Доказателство: [3] Нека $\langle V_a, \overline{V}_a\rangle$ е i-тия разрез в графа G с n върха и m дъги, а s_i е съответния вектор на разреза. Нека

$$V_a = \{v_1, v_2, ..., v_r\}, \text{ а } \overline{V}_a = \{v_{r+1}, ..., v_n\}$$

и a_i $(1 \le i \le n)$ е вектора в матрицата на инцидентност, съответстващ на v_i.

Без ограничение на общността да допуснем, че ориентацията на разреза е от V_a към $\overline{V}_a$. Ще докажем, че:

$$(4.7) \qquad s_i = a_1 + a_2 + ... + a_r = -(a_{r+1} + a_{r+2} + ... + a_n).$$

Нека v_p и v_s са върхове инцидентни с k-тата дъга, ориентирана от v_p към v_s. Следователно

$$(4.8) \qquad a_{pk} = 1, \quad a_{sk} = -1, \quad a_{jk} = 0, \quad j \neq p, \ k.$$

Случай 1. $v_p \in V_a$, а $v_s \in \overline{V}_a$, т.е. $p \le r$, а $s \ge r + 1$, $\Longrightarrow s_{ik} = 1$.

Случай 2. $v_p \in \overline{V}_a$, а $v_s \in V_a$, т.е. $p \ge r+1$, а $s \le r$, $\Longrightarrow s_{ik} = -1$.

Случай 3. $v_p, v_s \in V_a$, т.е. $p, \ s \le r \Longrightarrow s_{ik} = 0$.

Случай 4. $v_p, v_s \in \overline{V}_a$, т.е. $p, \ s \ge r + 1 \Longrightarrow s_{ik} = 0$.

Във всеки от възможните четири случая от (4.8) следва

$$(4.9) \qquad s_{ik} = (a_{1k} + a_{2k} + ... + a_{rk}) = -(a_{r+1,k} + a_{r+2,k} + ... + a_{nk}).$$

Тъй като (4.9) е вярно за $1 \le k \le m$, то следва верността на (4.7), което доказва и верността на теоремата. ◁

СЛЕДСТВИЕ 1.

$$rank(S) \leq rank(A_I).$$

СЛЕДСТВИЕ 2.

$$rank(S) = rank(A_I).$$

Последното е очевидно, като се вземе предвид, че матрицата на инцидентност A_I е подматрица на матрицата на разрезите S, т.е. $rank(S) \geq rank(A_I)$ и доказаното следствие 1.

От т.2 и т.3 на теорема 4.1 и следствие 2 на теорема 4.8, следва верността на твърдението:

▷ *ТЕОРЕМА 4.9. Нека G е граф с n върха и k компоненти.*

$$rank(S) = n - k$$

(рангът на матрицата на разрезите съвпада с ранга на графа). ◁

В свързания граф G с n върха покриващото дърво T определя множество от $n-1$ фундаментални (базисни) разрязващи множества. Да припомним, че всяко от тях съдържа едно и само едно ребро (дъга), принадлежащо на дървото T.

Подматрицата S_f на матрицата S, съответстваща на това множество, се нарича *фундаментална (базисна) матрица на разрязващите множества* относно дървото T, т.е. матрицата S_f има $n-1$ реда, съответни на фундаменталните разрези и m стълба (m е броят на ребрата (дъгите) в G). Нейните елементи s_{ij} са 1, когато реброто (дъгата) e_j принадлежи на i-тия фундаментален разрез и 0, в противен случай.

Ясно е, че ако ребрата (дъгите) непринадлежащи на дървото T номерираме от 1 до $\mu(G)$, а ребрата (дъгите) от дървото T номерираме от $\mu(G)+1$ до m, матрицата на фундаменталните разрези S_f може да се представи по следния начин:

$$S_f = (S_{11}|I), \tag{4.10}$$

където I е единична матрица от ред $n-1$, чиито стълбове съответстват на ребрата (дъгите) от дървото T.

Очевидно е, че с подходяща номерация S_{11} и I могат да си сменят местата. Когато графът е свързан, ориентацията на базисното разрязващо множество се избира така, че да съответства на ориентацията на определящата дъга от дървото.

От (4.10) следва, че $rank(S_f) = n - 1$, т.е. равен на ранга на матрицата S. С други думи всеки вектор-разрез може да се изрази като линейна комбинация от базисните вектори-разрези.

За графа от пример 4.6 матрицата на фундаменталните разрези, съответна на дървото $T = \{e_1, e_2, e_3, e_4, e_5\}$ е:

$$
(4.11) \qquad S_f = \begin{array}{l} \\ \\ \text{Фундаментален разрез 1} \\ \text{Фундаментален разрез 2} \\ \text{Фундаментален разрез 3} \\ \text{Фундаментален разрез 4} \\ \text{Фундаментален разрез 5} \end{array}
\begin{array}{c} \begin{array}{cccccc} e'_1 & e'_2 & e'_3 & e'_4 & e'_5 & e'_6 \\ e_6 & e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\
\left[\begin{array}{c|ccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} .
$$

В сила са следните интересни релации между матриците на инцидентност, на циклите и на разрезите. Ако разглеждаме неориентирани графи без примки и всички аритметични операции са операции по $mod\ 2$, са в сила следните 2 твърдения:

▷ *ТЕОРЕМА 4.10. Матрицата на инцидентност A_I и транспонираната матрица на фундаменталните цикли C_f^t са ортогонални, т.е.*

$$A_I.C_f^t = 0.$$

Доказателство: Верността на тази теорема следва директно от факта, че всеки връх в цикъла е инцидентен с четен брой ребра от този цикъл (ако цикълът е прост — с две ребра) и това, че операциите са по $mod\ 2$. ◁

▷ *ТЕОРЕМА 4.11. Матрицата на фундаменталните цикли C_f и транспонираната матрица на фундаменталните разрези S_f^t са ортогонални, т.е.*

$$C_f.S_f^t = 0.$$

Доказателство: Верността на теоремата следва от факта, че всеки разрез на един цикъл, индуциран от разрез в графа, се състои от четен брой ребра на разреза и освен това всички операции са по $mod\ 2$. ◁

Последните две теореми са верни и без да се използва предположението за фундаменталност на циклите и разрезите, т.е. верни са за произволни матрици на циклите и разрезите.

Освен това, тези две теореми имат "алгоритмичен" характер. От теорема 4.11 например следва, че

$$C_f.S_f^t = (I|C_{12}).\left(\frac{S_{11}^t}{I}\right) = S_{11}^t + C_{12} = 0.$$

Оттук $S_{11}^t = -C_{12}$. Но $-1 = 1(mod\ 2)$, следователно

$$S_{11}^t = C_{12}. \tag{4.12}$$

С други думи, ако е известна матрицата на фундаменталните разрези, веднага може да се получи с (4.12) матрицата на фундаменталните цикли и обратно.

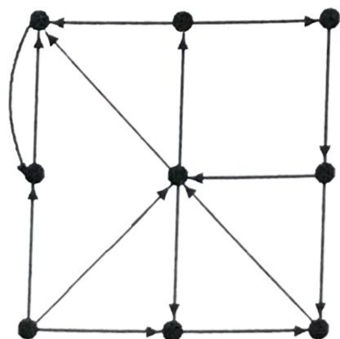
Например, за неориентирания граф от пример 4.5 матрицата на фундаменталните цикли C_f намерихме — виж (4.6). Тогава от (4.12) веднага получаваме за този граф матрицата на фундаменталните разрези:

$$S_f = (S_{11}|I) = \left[\begin{array}{cccc|ccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right].$$

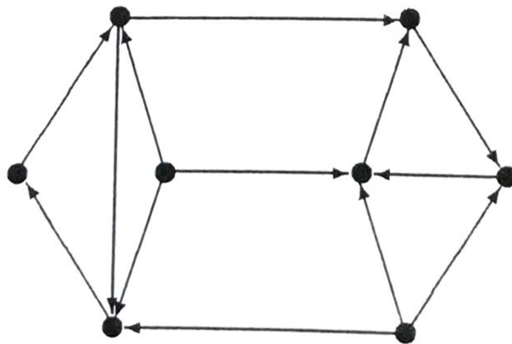
Аналогично, ако използваме (4.12) и намерената матрица на фундаменталните разрези (4.11) за графа от пример 4.6 веднага можем да получим матрицата на фундаменталните цикли за този граф:

$$C_f = (I|C_{12}) = [1|1\ 1\ 1\ 1\ 1].$$

ЗАДАЧА 4.1. За графите G_1 и G_2 от черт. 1.15 да се намерят:



Граф G_1



Граф G_2

Черт. 1.15

а) всички матрици, разглеждани в този параграф;
- матрица на инцидентност A_I;
- матрица на съседство B;
- матрица на достижимост R;
- матрица на контрадостижимост Q;
- матрица на циклите C;
- матрица на фундаменталните цикли C_f;
- матрица на разрезите S;
- матрица на фундаменталните разрези S_f;

б) броя на покриващите дървета в графа и самите дървета;

в) силно свързаните компоненти на графа и базите.

ЗАДАЧА 4.2. Да се определи кои от следните числови последователности са матрици (вектор) на степените на прост неориентиран граф:

а) $D_1 = (3, 3, 2, 2, 6, 3, 3, 3, 3)$; б) $D_2 = (2, 4, 3, 4, 3, 3, 3, 4)$;
в) $D_3 = (6, 5, 4, 3, 2, 1)$; г) $D_4 = (6, 4, 3, 1, 1, 1)$.

Отг. а) и б).

1.5. K-свързани графи. Сдвоявания. Покрития

Във втори параграф на тази глава беше въведено понятието свързаност — графът G е свързан, ако между всеки два негови върха съществува път. В следващите параграфи 3 и 4 бяха изяснени редица свойства на свързаните графи.

Водени от желанието да отговорим на въпроса "колко добре" е свързан един граф, в този параграф ще дадем някои резултати, касаещи така наречената k-свързаност.

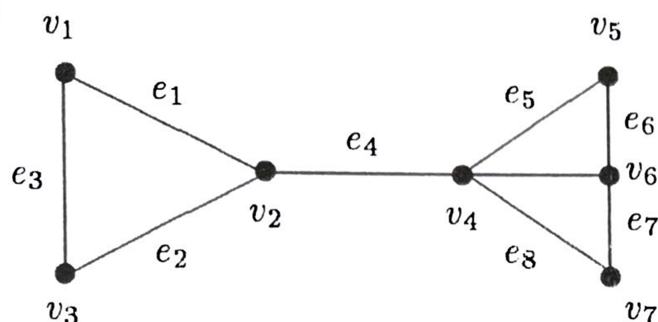
Ще формулираме и дадем доказателства на някои класически теореми от теория на графите — теорема на Менгер, теорема на Хол (Hall), теорема на Берж (Berge) и др., на които се базират много алгоритми за решаване на реални практически задачи.

Ще бъде въведено и изследвано понятието сдвояване (паросочетание (руск.); matching, assignment (англ.)) и понятието *биполярен (двухроматичен) граф*.

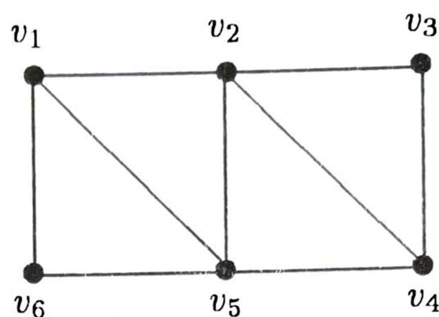
1. Ребрена и върхова k-свързаност

Да разгледаме следните графи:

(5.1)



Граф G_1



Граф G_2

Множествата от ребра $\{e_4\}$, $\{e_1, e_2\}$, $\{e_5, e_8, e_9\}$ са прости разрязващи множества — всяко от тях е минимално множество ребра, чието отстраняване нарушава свързаността на графа.

Броят ребра в разреза с минимално число ребра се бележи с $æ'(G)$ и се нарича *ребрена свързаност* $æ'(G)$.

С други думи $æ'(G)$ е минималния брой ребра, чието отстраняване от графа го превръща в несвързан или тривиален граф (припомняме, че при отстраняването на ребро, неговите крайни върхове не се отстраняват от G).

За графа G_1 от (5.1), $æ'(G_1) = 1$, а $æ'(G_2) = 2$.

	Графът G се нарича *k-ребрено-свързан*, ако $æ'(G) \geq k$.

Величината $æ'$ е мярка за свързаност.

От даденото определение следва, че графът G_1 е 1-ребрено-свързан, а графът G_2 е 2-ребрено-свързан (той е и 1-ребрено-свързан). Очевидно несвързаният граф е 0-ребрено-свързан.

От друга страна във всеки граф G съществува минимален брой върхове, чието отстраняване превръща графа в несвързан или тривиален. Този минимален брой върхове се бележи с $æ(G)$ и се нарича *върхова свързаност* на графа. Припомняме, че при отстраняването на връх от графа се отстраняват и инцидентните с него ребра.

Графът G_2 от (5.1) остава свързан при отстраняване на произволен негов връх. Отстраняването на върховете v_2 и v_4 води до несвързан граф, т.е. върховата свързаност на графа G_2 е 2, т.е. $æ(G_2) = 2$.

Очевидно върховата свързаност на несвързан граф е 0. Величината $æ(G)$ също е мярка за свързаност.

	Един граф G се нарича *k-свързан*, ако $\text{æ}(G) \geq k$.

Множеството върхове, чието отстраняване превръща графа в несвързан или тривиален, понякога се нарича *разделящо множество* в графа G. С други думи, k-свързаният граф не съдържа разделящи множества с мощност $\leq k-1$.

Според дадените определения, всеки свързан граф е поне 1-свързан и освен това, ако свързаният граф не притежава свързващи точки, то неговата свързаност е > 1 (виж началото на параграф 1.3, където е дефинирано понятието свързваща точка). Следователно свързаните графи без свързващи точки са поне 2-свързани. Несвързаният граф е 0-свързан.

ЗАДАЧА 5.1. Ако K_n е пълен граф с n върха, да се определи $\text{æ}(K_n)$.

Отг: $\text{æ}(K_n) = n - 1$.

ЗАДАЧА 5.2. Ако графът G не е пълен, да се определи горна граница за $\text{æ}(G)$. *Отг:* $\text{æ}(G) \leq n - 2$.

ЗАДАЧА 5.3. Да се конструира граф G, за който $\text{æ}(G) < \text{æ}'(G)$.

Ще формулираме и докажем някои твърдения, отнасящи се до $\text{æ}(G)$, $\text{æ}'(G)$ и $\delta(G)$, където $\delta(G)$ е минималната от степените на върховете.

▷ *ТЕОРЕМА 5.1. Ако G е прост свързан граф, то*

$$\text{æ}(G) \leq \delta(G). \tag{5.2}$$

Доказателство: Нека v_i е произволен връх на графа G. Множеството върхове, съседни с v_i, т.е. множеството $\Gamma(v_i)$ е очевидно разделящо множество — отстраняването на върховете от $\Gamma(v_i)$ изолира върха v_i или превръща графа в тривиален. Следователно за $\forall v_i \in V$

$$\text{æ}(G) \leq |\Gamma(v_i)| = d(v_i) \implies$$

$$\text{æ}(G) \leq \min_{v_i \in V}\{d(v_i)\} = \delta(G).$$

◁

▷ *ТЕОРЕМА 5.2. Ако G е прост свързан граф с n върха и m ребра, то*

$$\text{æ}(G) \le \left[\frac{2m}{n}\right], \tag{5.3}$$

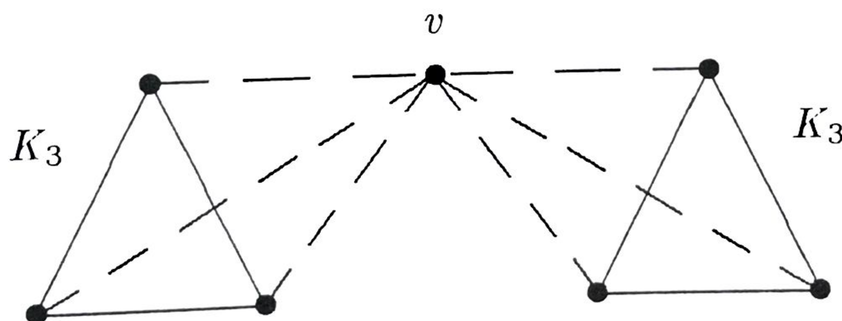
където $[x]$ е цялата част на x, т.е. най-голямото цяло число $\le x$.

Доказателство: От теорема 1.1 имаме

$$d(v_1) + d(v_2) + \dots + d(v_n) = 2m.$$

Следователно $n\delta(G) \le 2m$. Оттук и теорема 5.1 следва (5.3). ◁

За да се отстрани лъжливото впечатление, че върховата и ребрената свързаност са равни (виж графите G_1 и G_2 от (5.1)), ще конструираме граф G по следния начин: обединяват се два непресичащи се пълни графа K_n и се добавя нов връх v, свързан с всички останали върхове. Например за графа по-долу, получен по този начин, имаме:



$\text{æ}(G) = 1$ - множеството $\{v\}$ е разделящо;
$\text{æ}'(G) = 3$ - (в общия случай $\text{æ}'(G) = n$).

▷ *ТЕОРЕМА 5.3. Ако G е прост граф, то*

$$\text{æ}(G) \le \text{æ}'(G) \le \delta(G). \tag{5.4}$$

Доказателство: [3] Тъй като ребрата, инцидентни с произволен връх v в графа G образуват разрез, то за напълно произволен граф очевидно

$$\text{æ}'(G) \le \delta(G).$$

Ще докажем първото от неравенствата (5.4).

А) Ако графът G не е свързан, верността на (5.4) е тривиална, поради

$$æ(G) = æ'(G) = 0.$$

B1) Ако графът е свързан и $æ'(G) = 1$, в графа ще съществува ребро e, чието отстраняване води до увеличаване броя на компонентите (ребро с това свойство се нарича *мост*; виж графа G_1 от (5.1), реброто e_4 е мост). Отстраняването на връх, инцидентен с моста, води до граф, който е несвързан или тривиален, следователно

$$æ(G) = æ'(G) = 1,$$

т.е. в сила е (5.4).

B2) Нека графът G е свързан и $æ'(G) \geq 2$.

1. В графа съществуват $æ'(G)$ ребра, чието отстраняване води до несвързаност.

2. Отстраняването на всеки $æ'(G) - 1$ от тези ребра води до граф с мост $e = (v_1, v_2)$.

3. За всяко от тези $æ'(G) - 1$ ребра можем да изберем краен връх, различен от v_1 и v_2 (G е прост).

4. Отстраняването на всеки такъв краен връх (тези върхове са $æ'(G) - 1$) води до отстраняването на поне $æ'(G) - 1$ ребра.

Нека след прилагането на 4 се получава граф:

а) който е несвързан. Тогава

$$æ(G) \leq æ'(G) - 1;$$

б) който е свързан. Тогава в този граф ще има мост e и отстраняването на върха v_1 или v_2 на моста що води до несвързан или тривиален граф. Следователно

$$æ(G) \leq æ'(G).$$

И така във всички възможни случаи е изпълнено първото неравенство на (5.4). ◁

ЗАДАЧА 5.4. Нека G е прост граф с n върха. Ако

$$\delta(G) \geq [n/2], \text{ то } æ'(G) = \delta(G),$$

където $[x]$ е цялата част на x.

Упътване: Покажете, че G е свързан граф, откъдето следва $æ'(G) > 0$. От теорема 5.3 следва, че е достатъчно да се покаже, че $æ'(G) \geq \delta(G)$.

Сега ще формулираме един резултат, получен от Едмондс, като ще дадем доказателството на този резултат, предложено в [18] от Wang и Kleitman.

▷ *ТЕОРЕМА 5.4. Нека* $D = (d(v_1), d(v_2), ..., d(v_n))$ *е последователност от степените на върховете на граф* G. *Необходимо и достатъчно условие простият граф* G *да бъде* k*-ребрено-свързан* $(k \geq 2)$ *е за* $\forall i$, $d(v_i) \geq k$.

Доказателство: *Необходимостта* е очевидна поради това, че ако допуснем съществуването на връх със степен $< k$, то отстраняването на инцидентните с този връх ребра (а те са $< k$) прави графа несвързан, което е невъзможно поради k-ребрената свързаност на графа.

Достатъчност: С индукция ще покажем, че алгоритъмът, който описахме в предишния параграф 1.4, генерира при $d(v_i) \geq k$, k-ребрено-свързан граф.

Да направим индуктивно допускане, че алгоритъмът "работи" при $d(v_i) \geq p$ за $\forall i$ и $p \leq k - 1$.

Ще докажем, че в графа, строен с помощта на алгоритъма, всяко разрязващо множество $\langle A, \overline{A} \rangle$ съдържа най-малко k ребра.

Ако $|A| = 1$ или $|\overline{A}| = 1$, верността е очевидна.

Нека $|A| \geq 2$ и $|\overline{A}| \geq 2$. При изпълнение на алгоритъма, на стъпка r (стъпката, на която r-тия връх е напълно свързан) са възможни следните три случая, които напълно изчерпват възможните варианти.

Случай 1. Всички ненулеви остатъчни степени са не по-малки от k.

Случай 2. Всички ненулеви остатъчни степени са не по-малки от $k - 1$ и съществува поне едно ребро, построено на стъпки 1, 2, ..., r, съдържащо се в $\langle A, \overline{A} \rangle$.

Случай 3. Всички ненулеви остатъчни степени са не по-малки от $k - 2$ и съществуват поне две ребра, построени на стъпки 1, 2, ..., r, съдържащи се в $\langle A, \overline{A} \rangle$.

От индуктивното предположение, разрязващото множество $\langle A, \overline{A} \rangle$ ще съдържа не по-малко от k ребра във всеки от тези три случая. Остава да покажем, че наистина изброените горе случаи описват всички възможни варианти.

Нека върхът v_i се съединява с други върхове на стъпка i и без ограничение на общността нека върхът $v_1 \in A$. Ще покажем, че ако случаите 1 и 2 не възникват, то на някоя стъпка от алгоритъма възниква случай 3.

На стъпка 1 върхът v_1 е напълно свързан (занулена е степента на върха). Тогава:

а) най-малките ненулеви остатъчни степени са равни на $k-1$, тъй като случай 1 не е възникнал;

б) степените на нито един от върховете A не са намалени с 1 при съединяване с върха v_1, тъй като случай 2 не е възникнал.

Следователно върхът $v_2 \in A$ (всички върхове от A имат степен, не по-малка от k). Ако след съединяването на върха v_2 не възникне случай 1 и никое ребро не свързва A с $\overline{A}$, тогава все още, както и преди, ще имаме случай а) или б). Следователно следващият връх, който ще бъде свързван, ще бъде отново в A.

На всяка стъпка от алгоритъма степените намаляват и рано или късно ще бъде свързан връх $v_r \in A$, при което ще се появи ребро между A и $\overline{A}$. Ако не възникне случай 2, то в A ще има връх, който още не е напълно свързан и остатъчната му степен ще стане $k-2$. Това означава, че върхът v_r трябва да се съедини с всички върхове от $\overline{A}$, тъй като всички върхове от $\overline{A}$ имат остатъчни степени, равни на k, а ние съединяваме върха v_r с върховете, имащи най-големи остатъчни степени. Тъй като $|A| \geq 2$, на тази стъпка възниква случай 3.

С това доказателството е завършено. ◁

В [8] Wang и Kleitman дават необходими и достатъчни условия последователността $D = (d(v_1), d(v_2), ..., d(v_n))$ да бъде последователност от степени на прост k-(върхово) свързан граф.

2. Теорема на Менгер

Ще формулираме и докажем един класически резултат от теория на графите (теория на свързаността) — теорема на Менгер. Доказана е през 1927 г. в [19] и е аналог на теоремата за максималния поток и минималния разрез, както ще се убедим в следващата глава. Теоремата на Менгер може да се докаже (разглежда) като следствие от теоремата на Форд-Фалкерсон за максималния поток и минималния разрез и обратно. Ето защо ще дадем и директно доказателство на теоремата на Менгер, направено от Болобаш в [12].

Два $(s-t)$ пътя се наричат *независими пътища*, ако единствените им общи върхове са s и t. Понякога вместо независими,

такива $(s-t)$ пътища се наричат *върхово непресичащи се пътища.*

Ако W е множество от върхове или ребра в свързания граф G, а графът $G-W$ е несвързан, казваме, че W *разделя* G. Ако върховете s и t принадлежат на различни компоненти в $G-W$, казваме, че W *разделя* s и t.

▷ *ТЕОРЕМА 5.5. (Теорема на Менгер за върховете) Нека s и t са различни несъседни върхове на графа G. Тогава минималният брой върхове, които разделят s и t, е равен на максималния брой независими $(s-t)$-пътища.* ◁

▷ *ТЕОРЕМА 5.6. (Теорема на Менгер за ребрата) Нека s и t са различни върхове на графа G. Тогава минималният брой ребра, които разделят s и t, е равен на максималния брой $(s-t)$-пътища без общи ребра.* ◁

Втората от горните две теореми лесно се извежда от първата, затова ще докажем само теорема 5.5.

Доказателство: Нека k е минималният брой върхове, които разделят s и t. Тогава съществуват $\leq k$ независими $(s-t)$ пътя.

При $k \leq 1$ очевидно пътищата са точно k.

Нека сега $k \geq 2$.

$(*)$ Да допуснем, че теоремата не е в сила и нека за възможно най-малкото k, G е контрапример с минимален брой ребра.

Тогава ще съществуват $\leq k-1$ независими $(s-t)$ пътища и няма връх v, съединен едновременно със s и t (в противен случай $G-v$ ще бъде контрапример за $k-1$ $(**)$.

Да означим с W произволно множество от k върха, разделящо s от t. Ще докажем, че някой от върховете s и t е съседен с всички върхове от W. Да допуснем противното, т.е. s и t не са съседни с всички върхове от W. Тогава да разгледаме графа G_s, получен от G чрез "свиване" на компонентата $G-W$ до връх s', който е свързан с всички върхове на W. В G_s за разделянето на s' и t също са необходими k върха. Тъй като в $G-W$ има поне два върха, то G_s има по-малко ребра, отколкото в G. В G_s съществуват k независими $(s'-t)$ пътя, тъй като G е контрапример с минимален брой ребра. Частите на тези пътища от t до W имат точно един общ връх и той е t, т.е. точно един от тези пътища е $(t-w)$ път, $w \in W$.

Аналогично, можем да получим k пътя от s до W, всеки два от които имат точно един общ връх и той е s, т.е. точно един от тези пътища е $(s-w)$ път, $w \in W$. Ясно е, че ако обединим

пътя $(s - w)$ с пътя $(w - t)$, ще получим $(s - t)$ път. С други думи, получаваме k на брой независими $(s - t)$ пътища, което е противоречие.

И така, върхът s или върхът t е съседен с всички върхове от произволното множество W с k върха, разделящо s и t.

Да разгледаме най-краткия (най-малко ребра) $(s - t)$ път

$$(s,\ v_1,\ v_2,\ \ldots,\ v_l,\ t).$$

В този път $l \geq 2$ (виж $(*)$ ... $(**)$). Поради минималността на G, в графа $G - (v_1, v_2)$ съществува множество W_0 с $k - 1$ върха, разделящо s и t. Множествата

$$W_1 = \{v_1\} \cup W_0 \quad \text{и} \quad W_2 = \{v_2\} \cup W_0$$

са k елементни множества, разделящи s и t. Тъй като t и v_1 не са съседни, s е съединен с всички върхове от W_1.

Аналогично, тъй като s и v_2 не са съседни, то t е съединен с всички върхове от W_2. Тъй като $W_1 \cap W_2 \neq \emptyset$, следва, че s и t имат поне един общ съсед. Такъв е всеки връх от W_0, а $|W_0| = k - 1 \geq 1$. Това противоречи на $(*)$ ... $(**)$. С това теоремата е доказана. ◁

От теореми 5.5 и 5.6 следва, че:

СЛЕДСТВИЕ 1. Графът G е k-свързан ($k \geq 2$) тогава и само тогава, когато има поне два върха и всеки два негови върха могат да бъдат съединени с k независими (върхово непресичащи се) пътища.

СЛЕДСТВИЕ 2. Графът G е k-ребрено-свързан ($k \geq 2$) тогава и само тогава, когато има поне два върха и всеки два негови върха могат да се съединят с k-ребрено-непресичащи се пътища.

3. Съчетания по двойки (Сдвоявания)

Ще разгледаме една на пръв поглед "несериозна" задача, която е емблематична за един много широк клас оптимизационни проблеми.

ЗАДАЧА 5.5. (Задача за сватбите) Имаме m момичета и n момчета. Знаем за всяко от момичетата кои момчета познава. При какви условия можем да омъжим всички момичета, като не стигаме дотам да омъжим момиче за момче, което не познава.

Разбира се, в горната задача се визира моногамния случай (никое момче не се жени за две или повече момичета и обратно). Задачата е интересна и в полигамния случай.

На езика на теория на графите задачата може да се преведе така. Даден е биполярен граф $G = (X, Y, A)$, в който върховете x_1, x_2, ..., x_m са съответни на всяко от момичетата, а върховете y_1, y_2, ..., y_n са съответни на всяко от момчетата. Реброто (x_i, y_j) е от графа тогава и само тогава, когато y_j е момче, което момичето x_i познава.

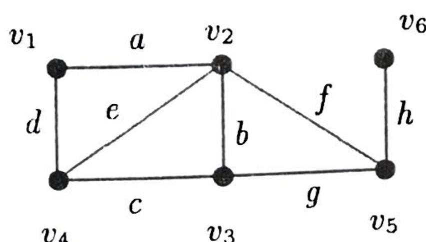
При какви условия можем да намерим такова множество от *независими ребра* (всеки две от тях нямат общ връх), което "насища" върховете на множеството X?

Всяко множество от независими ребра в графа се нарича *сдвояване* (съчетание по двойки).

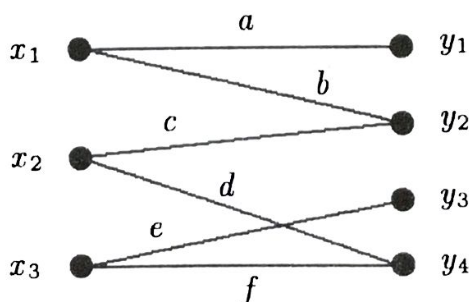
Сдвояването с най-голям брой ребра се нарича *максимално сдвояване*.

Върхът v се нарича *наситен* за сдвояването M, ако този връх се явява краен за ребро от M.

ПРИМЕР 5.1. Да разгледаме графите G_1 и G_2.



Граф G_1



Граф G_2

В графа G_1 множеството ребра $\{e,\ g\}$ е сдвояване, но то не е максимално. Максимално е например сдвояването $\{h,\ b,\ d\}$, както и сдвояването $\{h,\ c,\ a\}$.

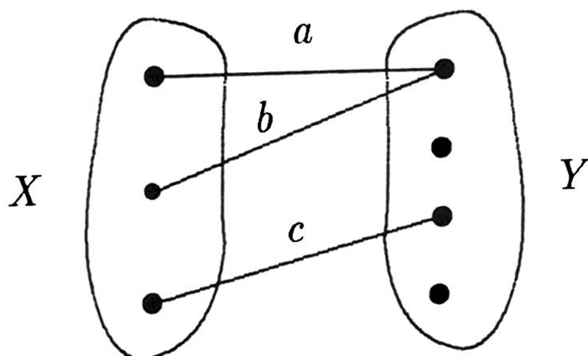
В биполярния граф G_2 от примера максималното сдвояване M е с мощност 3.

В графа G_1 върхът v_2 е наситен за сдвояването $M = \{e,\ g\}$, докато върхът v_1 е ненаситен.

Нека $G = (X, Y, A)$ е биполярен. Казваме, че X *се сдвоява с* Y, ако съществува сдвояване M, за което всеки връх на X е наситен в M. Такова сдвояване M се нарича *пълно сдвояване* на X с Y.

Например, в графа G_2 от примера, сдвояването $\{a,\ c,\ e\}$ е пълно.

От дефиницията е ясно, че пълното сдвояване в биполярния граф е максимално, но не всяко максимално сдвояване е пълно. Например, в биполярния граф G:



максималните сдвоявания (такива са $\{a,\ c\}$ и $\{b,\ c\}$) не са пълни сдвоявания на X с Y.

Сдвоявания, насищащи всички върхове на графа G, се наричат *съвършени (покриващи)*.

Такова е например сдвояването $\{h,\ b,\ d\}$ в графа G_1 от пример 5.1.

Цитираното вече сдвояване $\{a,\ c,\ e\}$ в графа G_2 от пример 5.1 е пълно (следователно и максимално), но то не е съвършено — върхът y_4 не е наситен.

ЗАДАЧА 5.6. Верни ли са за произволен граф G твърденията:

а) Всяко максимално сдвояване насища върховете на G;

б) Съществува максимално сдвояване, насищащо върховете на G.

Отг. Не.

Формулираната вече задача 5.5 за сватбите може да се модифицира и формулира в различна терминология. Например:

ЗАДАЧА 5.7. (Задача за играчките) Имате m на брой деца и n на брой играчки. Знае се за всяко от децата кои играчки харесва. Как да се разпределят играчките между децата (всяко дете получава една играчка), така че максимален брой деца да получат желана от тях играчка?

ЗАДАЧА 5.8. (Задача за назначенията) Дадени са m работника и n дейности. Знае се за всеки от работниците кои дейности може да извършва. По какъв начин да се разпределят дейностите (на всеки работник се възлага най-много една дейност), така че максимален брой работници да бъдат ангажирани.

Съществуват десетки интерпретации на така наречената задача за сватбите — максимален брой сделки, които трябва да осъществи посредник, получаващ комисион от всяка сделка; максимален брой игри, които могат да се осъществят в компютърен клуб и т.н. и т.н.

В този параграф ние ще разгледаме максимални сдвоявания, които са максимални по мощност (брой на участващите в тях ребра). В следващата глава на всяко ребро ще съпоставим тегло (коефициент на целесъобразност) и търсим сдвоявания, чието сумарно тегло е максимално. Ясно е, че в този случай няма да бъде задължително да търсим сдвояване с максимален брой ребра, тъй като то може да не е сдвояването с максимално сумарно тегло.

4. Сдвоявания в биполярни графи

Да се върнем сега отново на задача 5.5. Нека $G = (X, Y, A)$ е биполярният граф, съответен на задачата за сватбите и S е произволно множество от момичета, т.е. $S \subseteq X$. За множеството S и съответното му множество $\Gamma(S)$, което е подмножество на Y, има три възможности:

а) $|S| > |\Gamma(S)|$, т.е. $|S| - |\Gamma(S)| > 0$;

б) $|S| = |\Gamma(S)|$, т.е. $|S| - |\Gamma(S)| = 0$;

в) $|S| < |\Gamma(S)|$, т.е. $|S| - |\Gamma(S)| < 0$.

Очевидно в първия случай няма пълно сдвояване на S с $\Gamma(S)$ и всяко сдвояване M в графа ще насища не повече от $|\Gamma(S)|$ върха на S.

Да означим със $\sigma(S)$ разликата $|S| - |\Gamma(S)|$. Величината $\sigma(S)$ се нарича *дефицит на множеството* S [3].

От казаното следва, че за сдвояването M имаме

(5.5) $$|M| \leq |X| - \sigma(S).$$

Тъй като (5.5) е в сила за всяко $S \subseteq X$ следва, че за произволно сдвояване M имаме

(5.6) $$|M| \leq |X| - \max_{S \subseteq X}\{\sigma(S)\}.$$

Максималният от всички дефицити се нарича *дефицит на графа* G, т.е.

$$\sigma(G) = \max_{S \subseteq X}\{|S| - |\Gamma(S)|\}.$$

С други думи, (5.6) може да се запише

$$|M| \leq |X| - \sigma(G). \tag{5.7}$$

Обърнете внимание, че ако всички дефицити $\sigma(S)$ са по-малки или равни на 0 (случаите б) и в)), дефицитът на графа $\sigma(G) = 0$, тъй като за сдвояването $M = \emptyset$ имаме $|M| = 0$. С други думи, за произволен граф G дефицитът $\sigma(G) \geq 0$.

Ще формулираме сега теоремата на Хол:

▷ *ТЕОРЕМА 5.7. (Теорема на Hall) В биполярния граф* $G = (X, Y, A)$ *съществува пълно сдвояване тогава и само тогава, когато за* $\forall S \subseteq X$

$$|S| - |\Gamma(S)| \leq 0, \quad (\textit{т.е.}\ \ \sigma(G) = 0).$$

◁

Ще формулираме теоремата на Хол и на езика на сватбите:

▷ *ТЕОРЕМА 5.8. Задачата за сватбите има решение тогава и само тогава, когато всеки k момичета познават общо не по-малко от k момчета, $1 \leq k \leq m$, където m е броят на момичетата.*

Теорема 5.7 е доказана и от Халмош и Воган в [20], като идеята за тяхното доказателство е дадена от Болобаш в [12] с езика на сватбите (т.е. настоящата теорема).

Доказателство: *Необходимост:* Необходимостта е очевидна поради направените вече коментари (ако $|S| > |\Gamma(S)|$, следва, че не съществува пълно сдвояване).

Достатъчност: Да приложим индукция относно броя m на момичетата. При $m = 1$ твърдението е вярно. Да допуснем, че за $m \geq 2$ условието "всеки k момичета познават общо не по-малко от k момчета" е достатъчно условие за по-малки стойности от m.

1. Да допуснем, че всеки k момичета, $1 \leq k < m$, познават общо поне $k+1$ момчета. В такъв случай "уреждаме" една произволна сватба. Оставащите множества от момичета и момчета

удовлетворяват условието и според индуктивното допускане останалите $n-1$ момичета могат да бъдат омъжени.

2. Да допуснем, че за някое k има k момичета, които познават общо точно k момчета. В този случай да омъжим тези k момичета за тези k момчета. За оставащите множества от момичета и момчета условието на твърдението е изпълнено. Наистина, ако допуснем, че няки l неомъжени все още момичета познават по-малко от l на брой от останалите момчета, то споменатите $k+l$ момичета ще познават по-малко от $k+l$ момчета, което е невъзможно. Според индуктивното допускане оставащите момичета могат да бъдат омъжени за някои от оставащите момчета.

С това теоремата е доказана. ◁

Други по-формални доказателства на теоремата на Хол можете да намерите в [12], [3] и др.

СЛЕДСТВИЕ 1. За всеки еднороден биполярен граф съществува пълно сдвояване.

Верността директно следва от теоремата на Хол.

И така, от теоремата на Хол следва, че ако дефицитът на графа $\sigma(G)=0$, то в максималното сдвояване за биполярния граф $G=(X,Y,A)$ броят на ребрата е точно $|X|-\sigma(G)$, т.е. равенството в (5.7) се постига.

Може да се докаже, че и при $\sigma(G)>0$ броят на ребрата в максималното сдвояване е $|X|-\sigma(G)$. Достатъчно е да се добавят $\sigma(G)$ върха към Y и да ги съединим с всички върхове от X. Полученият по този начин нов граф удовлетворява условието за съществуване на пълно сдвояване. Това дава основание да формулираме следната *теорема на Кьониг*:

▷ *ТЕОРЕМА 5.9. Броят на ребрата в максималното сдвояване за биполярния граф $G=(X,Y,A)$ е равен на $|X|-\sigma(G)$, където $\sigma(G)$ е дефицитът на графа.* ◁

СЛЕДСТВИЕ 1. Можем да омъжим всичките момичета, с изключение на d от тях тогава и само тогава, когато които и да са k момичета познават общо поне $k-d$ момчета.

СЛЕДСТВИЕ 2. (Полигамия - i-тото момче възнамерява да се ожени за d_i момичета) Биполярният граф G съдържа подграф H, такъв, че $d_H(x_i)=d_i$ и $0\le d_H(y_j)\le 1$ тогава и само тогава, когато

$$\sum_{x_i\in S} d_i \le |\Gamma(S)| \quad \text{за } \forall S\subset X.$$

Верността на твърдението лесно се установява с помощта на теоремата на Хол. Заменете всеки връх x_i с d_i върха, съединени с всички върхове от $\Gamma(x_i)$. В новия граф ще съществува подграф H тогава и само тогава, когато има сдвояване на новия клас върхове X с върхове от Y.

СЛЕДСТВИЕ 3. Ако $G = (X, Y, A)$ е биполярен граф, за който

$$\min_{x \in X}\{d(x)\} \geq \max_{y \in Y}\{d(y)\},$$

то съществува пълно сдвояване на X с Y.

Доказателство: Нека $X_1 \subseteq X$. Да означим с A_1 множеството ребра, инцидентни с върховете X_1, а с A_2 — множеството ребра, инцидентни с $\Gamma(X_1)$. Тогава

$$|A_1| \geq |X_1| d_1 \quad \text{и} \quad |A_2| \leq |\Gamma(X_1)| d_2, \quad \text{където}$$

$$d_1 = \min_{x \in X}\{d(x)\}, \quad a \quad d_2 = \max_{y \in Y}\{d(y)\}.$$

Очевидно $A_1 \subseteq A_2$, ето защо

$$|X_1| d_1 \leq |A_1| \leq |A_2| \leq |\Gamma(X_1)| d_2. \tag{5.8}$$

По условие $d_1 \geq d_2$, следователно

$$\frac{1}{d_1} \leq \frac{1}{d_2}. \tag{5.9}$$

От (5.8) и (5.9) следва, че

$$|X_1| \leq |\Gamma(X_1)|, \quad \forall X_1 \subseteq X.$$

От теоремата на Хол следва, че съществува пълно сдвояване на X и Y. ◁

"Несериозната" задача за сватбите може да се формулира в "по-сериозна" терминология така. Нека M е непразно крайно множество, а $F = \{F_1,\ F_2,\ \dots,\ F_r\}$ е фамилия от непразни подмножества на M (не обезателно различни). *Система различни преставители на фамилията F (трансферзала)* се нарича множеството от r различни елементи на M по един от всяко F_i.

Например, ако

$$M = \{1,\ 2,\ 3,\ 4,\ 5\} \text{ и}$$

$$F_1 = \{1,\ 4,\ 5\},\quad F_2 = \{1,\ 4,\ 5\},\quad F_3 = \{1,\ 2,\ 4\},$$

то множеството $\{1,\ 4,\ 2\}$ е трансферзала на фамилията

$$F = \{F_1,\ F_2,\ F_3\}.$$

Ясно е, че не за всяка фамилия от подмножества, съществува система различни представители (трансферзала). Например, ако

$$F_1 = \{1,\ 5\} = F_2,\quad F_3 = \{5,\ 4\} \text{ и } F_4 = \{1,\ 4\},$$

то за тази фамилия няма трансферзала.

Ако поставим въпроса при какви условия за фамилията F съществува трансферзала, можем да направим следното. Да построим биполярния граф $G = (X,\ Y,\ A)$, в който:

1) върхът $x_i \in X$ е съответен на множеството F_i от фамилията F;

2) върхът $y_i \in Y$ съответства на елемента $i \in M$;

3) реброто $(x_i, y_j) \in A$, точно когато $j \in F_i$.

Оттук е ясно, че поставеният въпрос за съществуване на трансферзала е еквивалентен със задачата за намиране на пълно сдвояване на X и Y в горепостроения граф. Теоремата на Хол на езика на теория на трансферзалите може да се формулира така:

▷ *ТЕОРЕМА 5.10. Нека M е непразно множество и $F = \{F_1, F_2, \ldots, F_r\}$ е фамилия от негови подмножества. Фамилията F има трансферзала тогава и само тогава, когато обединението на произволни k $(1 \leq k \leq r)$ подмножества F_i съдържа не по-малко от k елемента на M.* ◁

Комбинаторно доказателство на тази теорема, без използване теория на графите, е направено от Радо в [21].

Матриците, имащи само нулеви и единични елементи, се наричат $(0,1)$-*матрици*. В такива матрици всеки ред или стълб се нарича още *линия*.

▷ *ТЕОРЕМА 5.11. (Кьониг - Егервари) Минималният брой линии, съдържащи всички "единици" на $(0,1)$-матрица е равен на максималния брой "единици", никои две от които не лежат на една линия в матрицата.*

Доказателство: [13] Да означим с A $(0,1)$-матрицата. Два елемента на A ще наричаме независими, когато не лежат на една и съща линия, т.е. a_{ij} и a_{kl} са независими тогава и само тогава, когато $i \neq k$ и $j \neq l$.

Да означим с m минималния брой линии, покриващи единиците на A, а с n максималния брой две по две независими единици. Очевидно $n \leq m$, защото ако допуснем противното, от принципа на Дирихле веднага ще стигнем до противоречие с "независимостта" на единиците. За да покажем равенството $n = m$, е достатъчно да покажем съществуването на система от m две по две независими единици.

Да направим произволно покритие на единиците в A с m линии, като при това тези m линии са всъщност p реда и q стълба, т.е. $p + q = m$. Без ограничение на общността можем да считаме, че всичките единици са разположени в първите p реда и q стълба (разместването на редове или стълбове не променя m или n). Да разгледаме множествата E_1, E_2, ..., E_p, където E_i е съставено от индексите j, за които $j > q$ и $a_{ij} = 1$.

Например за матрицата

$$p = 3 \qquad \left|\begin{array}{cccccc} 1 & 1 & \cdot & 0 & 1 & 0 & 1 \\ 0 & 1 & \cdot & 1 & 0 & 0 & 1 \\ 1 & 0 & \cdot & 1 & 0 & 1 & 1 \\ . & . & \cdot & . & . & . & . \\ 1 & 0 & \cdot & 0 & 0 & 0 & 0 \\ 0 & 1 & \cdot & 0 & 0 & 0 & 0 \end{array}\right|$$

$$q = 2$$

$$E_1 = \{4,\ 6\}, \quad E_2 = \{3,\ 6\}, \quad E_3 = \{3,\ 5,\ 6\}.$$

Ако j_1, j_2, ..., j_p е трасферзала (система от различни представители) на множествата E_1, E_2, ..., E_p, то очевидно a_{1j_1}, a_{2j_2}, ..., a_{pj_p} ще бъдат две по две независими единици, лежащи в първите p реда и вън от първите q стълба.

Множествата E_1, E_2, ..., E_p удовлетворяват теоремата на Хол (вж. теорема 5.10). Наистина, да допуснем противното, т.е. съществуват k индекса i_1, i_2, ..., i_k, $1 \leq k \leq p$, така че

$$E_{i_1} \cup E_{i_2} \cup \ldots E_{i_k}$$

съдържа не повече от $k - 1$ елемента. Следователно в редовете с номера i_1, i_2, ..., i_k се съдържат $\leq k - 1$ единици извън

първите q стълба. Тогава можем да построим ново покритие на единиците в A така: взимаме всички редове с номер i, $1 \leq i \leq p$ и $i \neq i_1, i_2, \ldots, i_k$, първите q стълба и още $k-1$ стълба, съдържащи тези $k-1$ единици. Построеното покритие съдържа $m-1$ линии, което противоречи на минималността на m.

Щом множествата $E_1, E_2, \ldots, E_p$ удовлетворяват теоремата на Хол, следва, че за тези множества съществува трансферзала или, което е същото, съществуват p независими единици, лежащи в първите p реда и извън първите q стълба.

Аналогично, ако разгледаме първите q стълба, ще получим q независими единици, лежащи в тях, които са вън от първите p реда.

Очевидно всяка единица от първата система е независима с всяка единица от втората система, поради което се получават общо $p + q = m$ две по две независими единици, с което доказателството е направено. ◁

Теоремата на Хол също може да се докаже като следствие от теоремата на Кьонинг.

Очевидно *теоремата на Кьонинг - Егервари* може да се перефразира по следния начин. Нека M е $(0,1)$ матрица с m реда и n стълба. Да построим биполярен граф $G = (X, Y, A)$ по следния начин:

1. върховете $x_i \in X$, $i = 1, 2, \ldots, m$, са съответни на редовете на матрицата M;

2. върховете $y_i \in Y$, $i = 1, 2, \ldots, n$, са съответни на стълбовете на матрицата M;

3. реброто $(x_i, y_i) \in A$, ако (i, j) елементът на M е равен на 1.

Ако разглеждаме върха като покриващ всички инцидентни с него ребра, *теоремата на Кьонинг - Егервари* може да формулираме така. "Минималният брой върхове в биполярния граф G, които покриват всички ребра, е равен на броя на ребрата в максималното сдвояване за графа." (вж. теорема 5.18.)

5. Сдвоявания в произволни графи

Ще анализираме въпроса за съществуване на максимални сдвоявания в произволни графи (не обезателно биполярни). За целта ще въведем някои допълнителни понятия.

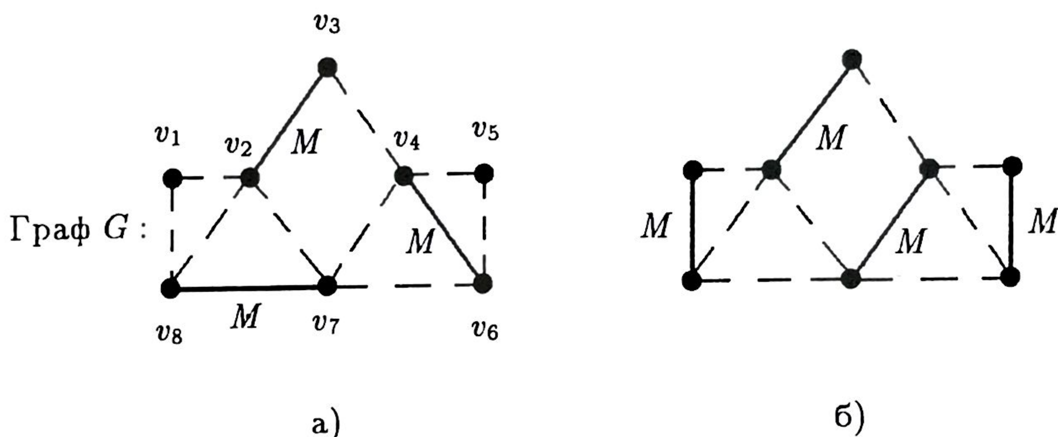
Ако $G = (V, A)$ е произволен граф и M е сдвояване в G, ребрата от сдвояването M ще наричаме *тъмни ребра* на графа за разлика от другите ребра, които ще наричаме *светли*.

Припомняме, че върховете инцидентни с ребра от M (т.е. тъмни ребра) наричаме *наситени върхове* за разлика от другите върхове, които наричаме *ненаситени върхове*.

Редуваща се (алтернативна) верига в графа G относно сдвояването M ще наричаме всяка верига от ребра, в която алтернативно участват тъмни и светли ребра.

Редуваща се верига относно сдвояването M, на която началния и крайния връх са ненаситени (експонирани), се нарича *аугментална (усилваща) верига*.

ПРИМЕР 5.2.



В графа G от подточка а) удебелените ребра са ребра на сдвояването M.

Наситените върхове относно това сдвояване са върховете v_8, v_7, v_2, v_3, v_4, v_6.

Тъмните ребра са маркирани с "M", пунктираните ребра са светли.

Веригата (v_8, v_2, v_3, v_4, v_6) е редуваща се (алтернативна) верига относно сдвояването M. Тази алтернативна верига не е аугментална (усилваща), тъй като единият от крайните й върхове е наситен (в случая и двата крайни върха са наситени).

Веригата (v_1, v_8, v_7, v_4, v_6, v_5) е аугментална верига, тъй като е алтернативна верига, на която крайните върхове v_1 и v_5 са ненаситени относно сдвояването M. Веригата е усилваща, защото ако в нея светлите ребра направим тъмни и обратно, тъмните — светли, ще получим ново сдвояване, в което броят на ребрата ще се увеличи с единица. В подточка а) сдвояването M е с мощност 3, а сдвояването M в подточка б) е с мощност 4.

Относно сдвояването M в подточка б) не съществуват усилващи вериги (всички върхове са наситени). Това сдвояване M, $|M| = 4$, е максимално за графа G. В един граф G може да съществуват различни (по състав на ребрата) максимални сдвоявания.

В сила е следното важно твърдение, доказано от Берж в [5]:

▷ *ТЕОРЕМА 5.12. (Теорема на Берж) Сдвояването M в графа G е максимално тогава и само тогава, когато в графа не съществуват усилващи относно M вериги.*

Доказателство: *Необходимост:* Нека сдвояването M_{max} е максимално, т.е. за всяко сдвояване M имаме

$$|M| \leq |M_{max}|. \tag{5.10}$$

Да допуснем, че относно M_{max} съществува усилваща верига P (с P означаваме веригата и множеството от нейните ребра) с крайни върхове v_1 и v_2, които са ненаситени. Ако светлите ребра от веригата P направим тъмни, а тъмните — светли, в графа G ще получим едно ново множество M от тъмни ребра. Това множество M от тъмни ребра очевидно е сдвояване в G, тъй като крайните върхове v_1 и v_2 са били ненаситени и M_{max} е сдвояване.

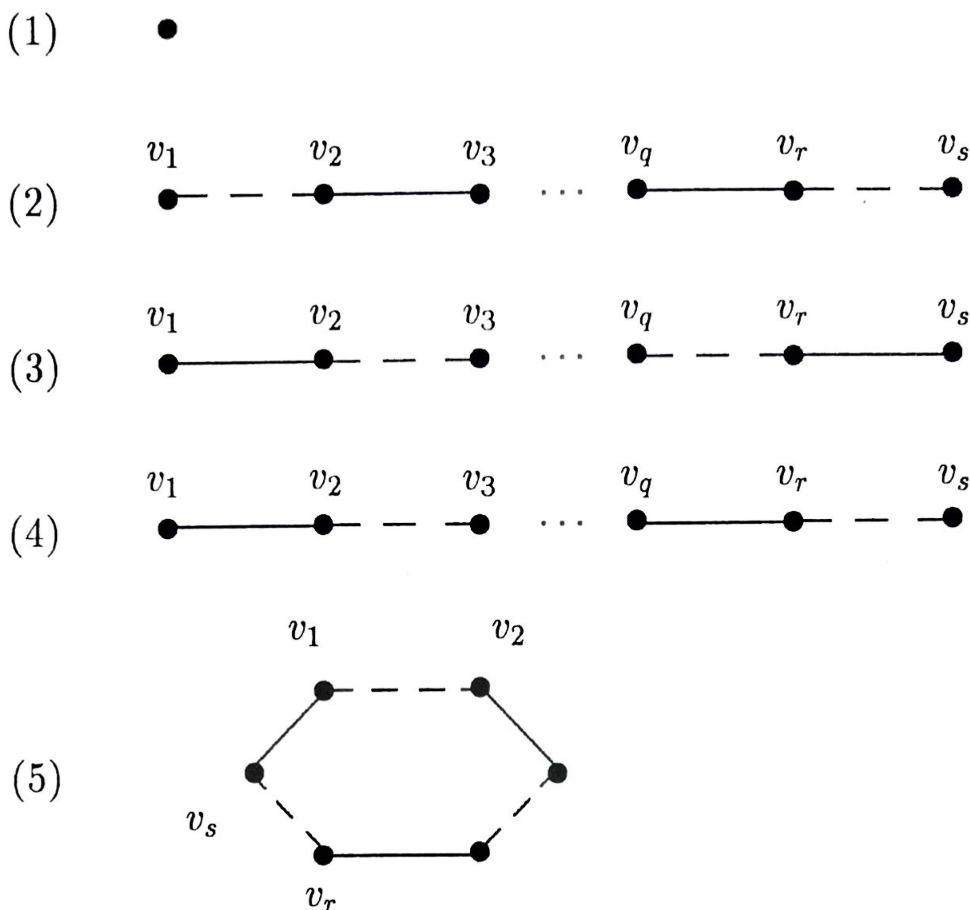
Освен това, очевидно за новото сдвояване M имаме

$$|M| = |M_{max}| + 1,$$

което противоречи на (5.10).

Достатъчност: Нека сега M е сдвояване, относно което не съществува увеличаваща верига в графа G и нека M_{max} е едно максимално сдвояване.

Да построим покриващ подграф G_p, състоящ се от ребрата $(M \cup M_{max}) - (M \cap M_{max})$ и всички върхове на G. Степените на всички върхове от G_p са ≤ 2 (допускането на противното води до противоречие с факта, че M и M_{max} са сдвоявания). Следователно подграфът G_p се състои от една или повече свързани компоненти, всяка от които е или изолиран връх, или проста верига, или прост цикъл, както е показано на чертежа долу:



Пунктираните ребра са от M_{max}, а другите от M.

Верига от тип (2) не съществува, тъй като е увеличаваща относно M верига, а такива по предположение няма.

Верига от тип (3) също не съществува, тъй като тя е увеличаваща относно M_{max}, което противоречи на максималността на M_{max} (виж необходимостта).

С други думи, не съществуват вериги от тип (2) и (3), в които крайните върхове са наситени в едно от сдвояванията.

Цикъл от типа (5) с нечетен брой ребра не съществува. Допускането на противното води до противоречие с това, че M и M_{max} са сдвоявания, т.е. никои две ребра в тези множества не са съседни.

Следователно, възможни са графи (компоненти) от типа (1), (4) и цикъл от типа (5) с четен брой ребра. За всеки такъв граф броят ребра $n(M)$ в M е равен на броя ребра $n(M_{max})$ в M_{max}. Тъй като това е вярно за всяка свързана компонента k на графа G_p, то

$$\sum_k n_k(M) = \sum_k n_k(M_{max}),$$

където $n_k(M)$ и $n_k(M_{max})$ са съответно броят на ребрата в ком-

понентата k, принадлежащи съответно на M и M_{max}. Следователно

$$|M| \equiv |M \cap M_{max}| + \sum_k n_k(M) = |M \cap M_{max}| + \sum_k n_k(M_{max}) \equiv |M_{max}|,$$

т.е. M се явява максимално сдвояване. ◁

Алгоритъмът за намиране на максимално сдвояване по мощност (брой ребра), който ще бъде разгледан в следващата глава, се базира на доказаната теорема на Берж.

ЗАДАЧА 5.9. Да се докаже, че в биполярния граф $G = (X, Y, A)$, чиято максимална степен на връх е Δ, т.е.

$$\max_{v \in X \cup Y} \{d(v)\} = \Delta$$

а) съществува сдвояване M_1, което насища всички върхове от X, чиято степен е Δ;

б) съществува сдвояване M_2, което насища всички върхове от Y, чиято степен е Δ.

Упътване: Виж следствие 3 от теорема 5.9.

ЗАДАЧА 5.10. (Теорема на Менделсон и Далмедж) Нека $G = (X, Y, A)$ е биполярен граф, а M_i е сдвояване на $X_i \subseteq X$ с $Y_i \subseteq Y$, $i = 1, 2$. Да се докаже, че съществува сдвояване $M' \subseteq M_1 \cup M_2$, което насища X_1 и Y_2.

Упътване: Разгледайте граф $G' = X_1 \cup X_2,\ Y_1 \cup Y_2,\ M_1 \cup M_2)$. От доказателството на теоремата на Берж следва, че всички компоненти на този граф са пътища или цикли от ребра, принадлежащи алтернативно на M_1 и M_2, т.е. степените на върховете на този граф са ≤ 2.

ЗАДАЧА 5.11. Да се докаже, че в биполярния граф $G = (X, Y, A)$ съществува сдвояване, насищащо всички върхове на графа с максимална степен.

Упътване: Верността е директно следствие от задачи 5.9 и 5.10.

ЗАДАЧА 5.12. Нека $G = (X, Y, A)$ е биполярен граф с максимална степен на връх Δ. Да се докаже, че множеството ребра на графа може да се разбие на Δ сдвоявания.

Решение: От задача 5.11 следва, че съществува сдвояване M_1, насищащо всички върхове от степен Δ. Очевидно в биполярия граф $G' = (X, Y, A - M_1)$ максималната степен е $\Delta - 1$. Пак поради задача 5.11 в него има сдвояване M_2, насищащо всички върхове от степен $\Delta - 1$ и т.н. докато се построи последователността от непразни сдвоявания $M_1,\ M_2,\ \ldots,\ M_\Delta$, образуващи разбиване на A.

Ще формулираме без доказателство теоремата на Тат (W. T. Tutte), даваща условия за съществуване на съвършено (покриващо) сдвояване в графа G.

Една компонента в графа $G = (V, A)$ се нарича *нечетна*, ако има нечетен брой върхове, и *четна* — в противен случай.

Ако $S \subseteq V$, то с $p_0(S)$ се бележи броят на нечетните компоненти в графа $G - S$.

▷ *ТЕОРЕМА 5.13. (Теорема на Тат) В графа $G = (V, A)$ има съвършено сдвояване тогава и само тогава, когато*

$$p_0(S) \leq |S|, \quad \text{за} \quad \forall S \subset V.$$

◁

6. Ребрени покрития в графи

В предишната подточка във връзка със задачата за назначенията и сватбите формулирахме два основни типа реални задачи:

	ЗАДАЧА (1). Намиране на максимално по мощност сдвояване M (максимален рой ребра);
	ЗАДАЧА (2). Намиране на сдвояване M с максимално тегло (максимално сумарно тегло на участващите в него ребра).

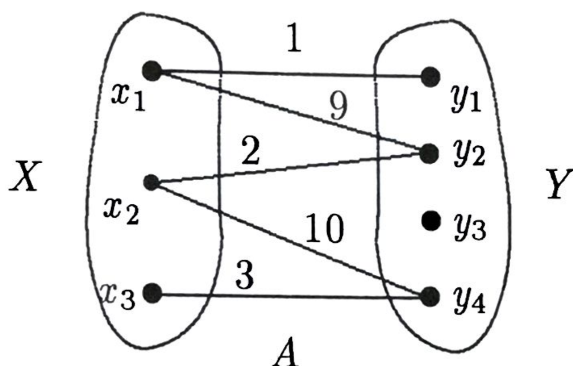
Ясно е, че задачите от горните два типа са различни, т.е. не всяко максимално по мощност сдвояване е максимално по тегло.

ПРИМЕР 5.3. Търговец на коли разполага със списък от клиенти, всеки от които иска да купи точно една кола. Търговецът знае за всеки от клиентите коя от наличните коли е склонен да купи.

ЗАДАЧА (1). Какъв е най-големият брой сделки, които може да реализира търговецът (търговецът получава един и същ комисион от всяка сделка)?

ЗАДАЧА (2). Ако търговецът знае теглото на своя комисион от всяка сделка (т.е. може да получава различни суми за различните сделки), кои сделки трябва да сключи, за да получи максимална печалба за себе си?

Например, на чертежа по-долу в биполярния граф $G = (X, Y, A)$ X е множеството от клиенти—купувачи, Y е множеството от коли за продан, а A е множеството от ребра, посочващи за всеки от клиентите коя кола е склонен да купува. Числата, приписани на всяко от ребрата A, са печалбата в стотици лева за търговеца при осъществяване на съответната сделка.



Очевидно, ако всички тегла бяха равни на единица, търговецът спокойно може да осъществи сделките (x_1, y_1), (x_2, y_2) и (x_3, y_4). Това е максималният брой сделки (максимално по мощност сдвояване).

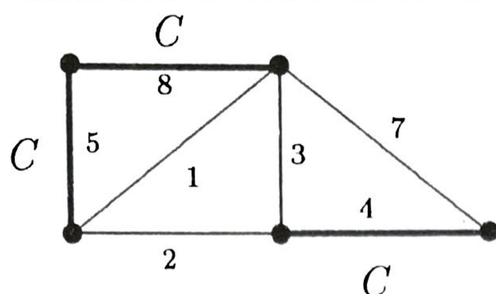
Тъй като обаче в конкретната ситуация различните сделки са с различна тежест, търговецът би получил профит, равен на $1 + 2 + 3 = 6$, ако реализира най-големия брой сделки. За него по-изгодно би било да реализира само две сделки, например (x_1, y_2), (x_2, y_4), които биха му донесли печалба $9 + 10 = 19$.

С други думи, *ЗАДАЧА (1)* и *ЗАДАЧА (2)* очевидно са различни и съдържателни. Разбира се, двата типа цитирани задачи могат да се разглеждат не само в биполярни, а в графи от общ тип.

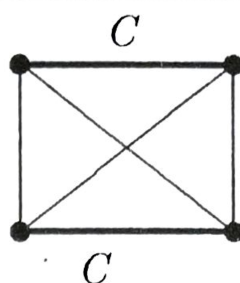
Ще въведем понятието ребрено покритие, свързано с други два типа (класа) оптимизационни задачи.

Нека $G = (V, A)$ е граф. Множеството ребра $C \subseteq A$ се нарича *ребрено покритие* на графа G, ако всеки връх на графа G е краен връх на ребро от C.

Например на чертежа долу са изобразени ребрени покрития на графите G_1 и G_2:



Граф G_1



Граф G_2

Покритието C в графа G_2 се явява и сдвояване, което не може да се каже за покритието C в графа G_1.

Очевидно множеството от всички ребра на графа се явява ребрено покритие на графа и е резонно да се търси минималния брой ребра, с които могат да се покрият върховете на графа.

Ребреното покритие с минимален брой ребра се нарича *минимално (по мощност) ребрено покритие.*

Ясно е, че ако ребрата на графа имат приписани тегла, може да се търси не минималното по брой ребра покритие, а покритието с минимално сумарно тегло. Такова покритие се нарича *ребрено покритие с минимално тегло.*

Например, в горе цитирания граф G_1 покритието C е минимално по мощност ребрено покритие, $|C| = 3$. Очевидно обаче това покритие не е ребрено покритие с минимално тегло, тъй като покритието, състоящо се от ребрата с тегла 5, 1, 2, 4 е покритие с тегло 12, докато минималното по мощност покритие C е с тегло 17. Не е трудно да се види, че цитираното покритие с тегло 12 не е покритие с минимално тегло, тъй като съществува покритие с тегло $5 + 1 + 4 = 10$.

ПРИМЕР 5.4. Налага се в югозападния регион на страната, състоящ се от 10 основни селищни системи, да се сформира комисия по опазване на околната среда. При това комисията трябва да включва поне по 1 представител от всяка селищна система и поне по 1 представител на всяка от 15-те парламентарно представени партии. Представени са за участие в тази комисия 120 кандидати. Да се определи съставът на комисията така, че да включва минимален брой членове, като са изпълнени споменатите изисквания.

Очевидно, ако построим граф, състоящ се от 10 върха съответни на селищните системи, плюс 15 върха съответни на всяка парламентарна партия, т.е. граф с 25 върха, в който всяка от 120-те кандидатури е ребро свързващо съответната селищна система, от която е кандидатът, с парламентарната партия, която кандидатът представлява, задачата се свежда до определяне на минимално по мощност ребрено покритие за построения граф.

Не е трудно да се съобрази, че ако на ребрата в графа съпоставим някакви тегла (напр. необходими бюджетни разходи за участие на съответния кандидат в комисията), можем да модифицираме задачата до задача за търсене на ребрено покритие с минимално (сумарно) тегло.

В резюме, налице са следните два нови класа реални оптимизационни задачи:

	ЗАДАЧА (3). Намиране на минимално по мощност покритие C (покритие с минимален брой ребра).
	ЗАДАЧА (4). Намиране на покритие C с минимално тегло.

Ще покажем, че редица други проблеми и задачи, които са съдържателни, лесно се свеждат до споменатите вече четири основни типа задачи.

А) ТЪРСЕНЕ НА СДВОЯВАНЕ С МИНИМАЛНА МОЩНОСТ:

Задачата е тривиална, тъй като сдвояването с нула ребра, т.е. $|M| = 0$ е сдвояване с минимална мощност.

Б) ТЪРСЕНЕ НА СДВОЯВАНЕ С МИНИМАЛНО ТЕГЛО:

Ако всички ребра са а с неотрицателно тегло, задачата отново е тривиална, тъй като сдвояването с нулева мощност е сдвояването с минимално тегло. Ако теглата на някои от ребрата са отрицателни (отрицателните разходи за една комуникация (ребро) могат да се интерпретират като печалба от тази комуникация), задачата лесно може да се реши така:

1. Изключват се от разглеждане ребрата, имащи неотрицателно тегло, а теглата, които са отрицателни, се заменят с противоположни стойности.

2. Намира се сдвояване с максимално тегло, т.е. *ЗАДАЧА (2)*. Ясно е, че ребрата в намереното сдвояване са сдвояване с минимално тегло за изходния граф. С други думи, задача Б) е еквивалентна на *ЗАДАЧА (2)*.

В) ТЪРСЕНЕ НА ПОКРИТИЕ С МАКСИМАЛНА МОЩНОСТ:

Както вече споменахме, задачата е тривиална, тъй като покритието, състоящо се от всчики ребра на графа (разглеждаме графи без изолирани върхове), представлява покритие с максимална мощност.

Г) ТЪРСЕНЕ НА ПОКРИТИЕ С МАКСИМАЛНО ТЕГЛО:

Очевидно в такова покритие ще участват всички ребра с поло-

жително тегло. Ако тези ребра не са достатъчни, за да покрият всички върхове на графа, то в това покритие трябва да се включат допълнително и някои ребра с неположително тегло, така че графът да се покрие. Очевидно, ако за непрокритите върхове потърсим покритие с минимално тегло, но след като сме обърнали знака на неположителните тегла, т.е. *ЗАДАЧА (4)*, ще осъществим оптимален избор на допълнителни ребра, гарантиращи получаване на покритието с максимално тегло. С други думи, Г) се свежда до *ЗАДАЧА (4)*.

Между основните *ЗАДАЧА (1)* и *ЗАДАЧА (3)*, т.е. между задачата за търсене на максимално по мощност сдвояване и задачата за търсене на минимално по мощност покритие, съществува връзка, която дава възможност решението на едната задача да генерира решение на другата и обратно.

Да означим $|M_{max}| = \alpha_1$ и $|C_{min}| = \beta_1$, т.е. α_1 е броя на ребрата в максималното по мощност ребрено сдвояване, а β_1 е броя на ребрата в минималното по мощност ребрено покритие.

В сила е следното твърдение:

▷ *ТЕОРЕМА 5.14. Ако $G = (V, A)$ е прост граф с n върха, който няма изолирани върхове, то $\alpha_1 + \beta_1 = n$.* ◁

Верността на това твърдение е следствие от разсъжденията, които ще направим по-долу. Да разгледаме следните две процедури:

Процедура 1. Нека M е произволно сдвояване в графа G.

Избираме произволен връх v, ненаситен от ребро на M. Добавяме към M произволно ребро, инцидентно с v.

Повтаряме тази процедура докато съществуват ненаситени върхове (т.е. за всички върхове v, които не са инцидентни с ребро от M).

Да означим с C^* полученото покритие в графа.

Процедура 2. Нека C е произволно покритие в графа G. Избираме произволен връх v, инцидентен с повече от едно ребро на C. Изключваме от C произволно ребро, инцидентно с такъв връх v.

Повтаряме процедурата, докато съществуват върхове v, инцидентни с повече от едно ребро на C.

Очевидно полученото в резултат на тази процедура множество от ребра M^* ще бъде сдвояване в графа G.

И така, с горните две процедури ние можем да преминаваме от произволно сдвояване M към покритие C^* (чрез добавяне

на ребра) и от произволно покритие C (чрез отстраняване на ребра) към сдвояване M^*.

Ще докажем, че ако сдвояаването M, от което стартира **процедура 1**, е максимално по мощност, т.е.

$$|M| = |M_{max}| = \alpha_1,$$

то получаваното в резултат на тази процедура покритие C^* е минимално по мощност покритие, т.е.

$$|C^*| = |C_{min}| = \beta_1.$$

Аналогично, ако покритието C, от което стартира **процедура 2**, е минимално по мощност покритие, то получаваното в резултат на тази процедура сдвояване M^* е максимално по мощност сдвояване, т.е. ако

$$|C| = |C_{min}| = \beta_1,$$

то за получаваното сдвояване M^* имаме

$$|M^*| = |M_{max}| = \alpha_1.$$

Доказателство: Нека M_{max} е произволно максимално сдвояване, т.е. $|M_{max}| = \alpha_1$. Очевидно сдвояването M_{max} има α_1 ребра и $2\alpha_1$ върха. Тъй като ненаситените в графа върхове са $n - 2\alpha_1$ и M_{max} е максимално сдвояване, ние ще получим с помощта на **процедура 1** покритие C^*, което ще се състои от $\alpha_1 + n - 2\alpha_1$ ребра, т.е.

$$|C^*| = n - \alpha_1, \quad \text{т.е.}$$

$$|M_{max}| + |C^*| = n. \tag{5.11}$$

Нека сега C_{min} е минимално покритие в графа, т.е. има β_1 ребра и естествено покрива n-та върха в графа. Ясно е, че ако приложим Δ пъти **процедура 2**, поради минималността на C_{min}, ние Δ пъти ще отнемаме по едно ребро и наситеност на точно един връх. По този начин ще получим сдвояване M^*, което ще има $n - \Delta$ върха и $\beta_1 - \Delta$ ребра. Но във всяко сдвояване, следователно и в M^*, броят на върховете е равен на два пъти броя на ребрата, т.е.

$$n - \Delta = 2(\beta_1 - \Delta);$$
$$n - \Delta = 2\beta_1 - 2\Delta;$$
$$n - \beta_1 = \beta_1 - \Delta;$$
$$n - \beta_1 = |M^*|; \quad \text{т.е.}$$

(5.12) $$|C_{min}| + |M^*| = n.$$

Равенствата (5.11) и (5.12) можем да запишем още така:

(5.13) $$\begin{aligned} \alpha_1 + \beta &= n, \\ \beta_1 + \alpha &= n, \end{aligned}$$

където с β сме означили $|C^*|$ и с α сме означили $|M^*|$. Ако извадим почленно равенствата (5.13), ще получим

(5.14) $$(\alpha_1 - \alpha) + (\beta - \beta_1) = 0.$$

Тъй като $\alpha_1 \geq \alpha$ и $\beta \geq \beta_1$, то от (5.14) следва:

(5.15) $$\alpha = \alpha_1 \quad \text{и} \quad \beta = \beta_1, \quad \text{т.е.}$$

получаваното чрез **процедура 1** покритие C^* е минимално и получаваното чрез **процедура 2** сдвояване M^* е максимално. От (5.13) следва, че

$$\alpha_1 + \beta_1 = n,$$

което всъщност искахме да докажем.

С други думи, ако ние знаем (разполагаме с) максимално по мощност сдвояване в графа G, лесно ще решим задачата за намиране на минимално по мощност покритие чрез **процедура 1**, както и обратната задача чрез **процедура 2**.

За съжаление, подобна връзка каквато установихме между *ЗАДАЧА (1)* и *ЗАДАЧА (3)*, не съществува между *ЗАДАЧА (2)* и *ЗАДАЧА (4)*, т.е. между задачата за сдвояване с максимално тегло и задачата за покритие с минимално тегло.

По-късно в следващата глава ще разгледаме алгоритми за решаване на *ЗАДАЧА (1)*, *ЗАДАЧА (2)* и *ЗАДАЧА (4)*, с което ще решим и редица други класове задачи, както стана ясно от направените коментари.

Освен това, решаването на цитираните горе три задачи се използва при реализилизането на алгоритми, търсещи Ойлерови вериги в графи и решаването на други проблеми не само в теория на графите.

ЗАДАЧА 5.13. Нека C_{min} е минимално ребрено покритие в G, състоящо се от r свързани компоненти. Да се докаже, че $|C_{min}| = \beta_1 = n - r$.

Упътване: В C_{min} не съществуват цикли и пътища с дължина по-голяма от 2. Следователно всяка компонента на покритието C_{min} е дърво, в което всички ребра са инцидентни с общ връх. От това директно следва $\beta_1 = n - r$.

7. Върхови покрития в графи

Ще разгледаме следните важни в теория на графите понятия, а именно:

Нека $G = (V, \Gamma)$ е граф. Подмножеството $S \subseteq V$ се нарича *независимо множество* в графа G, ако никои два върха в S не са съседни. Независимото множество S се нарича още *вътрешно устойчиво множество*. С други думи, независимото множество S е такова подмножество от върхове, за което

$$S \cap \Gamma(S) = \emptyset.$$

Независимото множество S се нарича *максимално независимо множество*, ако за всяко независимо множество S' е изпълнено $|S'| \leq |S|$.*

Броят върхове в максималното независимо множество на графа G се нарича *мярка за независимост (мярка за вътрешна устойчивост)* на графа G и се бележи с $\alpha_0(G)$ или само α_0 [5].

Въвеждането и използването на това понятие е обусловено от често възникващи задачи за търсене на такива подмножества от върхове, които притежават отнапред зададени свойства. В случая, търсене на максимално възможния брой върхове в графа, за които породения от тях подграф е напълно несвързан.

На практика съществуват "противоположни" реални задачи за намиране на максимален брой пунктове, всеки два от които са директно свързани, т.е. търсене на *максимален пълен подграф (клика)* в графа G, породен от някакво подмножество $S \subseteq V$. Тук максималността е в смисъл, че за всяко друго множество върхове $H \supset S$, породеният от H подграф не е пълен. Очевидно, кликата на практика не е нищо друго освен силно свързана компонента в графа, в която достижимостта е ограничена до пътища с единична дължина (директни пътища).

Кликовото число в графа G се нарича още *гъстота* или *плътност* и се определя по аналогия с α_0 като максималния брой върхове в кликите на дадения граф.

Очевидно в един "плътен" граф кликовото число грубо казано е голямо, а мярката за независимост е малка и обратно, т.е. между тези две понятия съществува така да се каже "противоположно" съотношение. Очевидно е и, че максимално независимото множество в G е съответно на кликата в $\overline{G}$ и обратно.

*В други литературни източници максималните независими множества се определят като независими множества несъдържащи се в никое друго независимо множество.

Съществува обаче още един клас задачи, в които се търси подмножество S от върхове с минимална мощност, така че всеки връх от множеството $V - S$ да е достижим от S директно, т.е. чрез път с единична дължина.

ПРИМЕР 5.5. Как да бъдат разположени телевизионни или други станции, медицински или други пунктове, контролиращи дадена територия, така че с минимален брой такива пунктове да бъде "покрита" територията? По-конкретно. Да разгледаме следната топографска карта от 16 квартала.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Какъв е минималният брой пунктове за "контрол" (медицински, полицейски, здравен и др.), които трябва да се разположат, ако се знае, че дислоцирането на контролен пункт в един от кварталите обслужва и съседните квартали, имащи обща граница с този квартал)?

С други думи, търсим минимален брой бази и тяхната локация с цел обезпечаване на контрол над цялата територия.

Тази задача очевидно прилича на задачата за търсене на база в граф, разгледана в параграф 4 на тази глава с тази разлика, че сега търсената достижимост е ограничена до пътища с единична дължина.

В конкретната задача от пример 5.5 е очевидно, че минималният брой контролни пунктове, които покриват цялата територия е 4 и възможни места за тяхното дислоциране са квартали с номера 2, 9, 15, 8 или 3, 5, 12, 14. Проблемът наистина е интересен, ако става въпрос за територии не от 16 квартала, а от 1600 квартала например, в които броят на съседните е различен.

От направения коментар е ясно, че е съдържателно и полезно да се въведе още едно понятие, а именно върхово покритие в графа $G = (V, \Gamma)$.

Подмножеството $K \subseteq V$ се нарича *върхово покритие* в графа G, ако всяко ребро в графа G има поне един краен връх в под-

множеството K. Върховото покритие K се нарича още *външно устойчиво множество* от върхове.

На други места в литературата то се нарича още *доминиращо множество от върхове* [5] и се дефинира като подмножество K, за което $K \cup \Gamma(K) = V$.

Ако всеки връх разгледаме като покриващ инцидентните с него ребра е ясно, че върховото покритие K в графа G е такова множество от върхове, което покрива всички ребра в графа.

Върховото покритие K в графа G е *минимално върхово покритие*, ако за всяко върхово покритие K' е изпълнено

$$|K'| \geq |K|,$$

т.е. минималното върхово покритие съдържа възможно наймалкия брой върхове. Броят върхове в минималното върхово покритие се нарича още *мярка за външна устойчивост* или *мярка за доминираност*. Бележи се с $\beta_0(G)$ или само β_0.

В предишната подточка доказахме връзката

$$\alpha_1 + \beta_1 = n,$$

т.е. връзката между мерките на максималното сдвояване и минималното ребрено покритие. Разглежданите сега мерки α_0 и β_0, отнасящи се до върхови покрития, също са свързани по аналогичен начин, а именно

$$\alpha_0 + \beta_0 = n.$$

Освен това $\alpha_1 \leq \beta_0$ и $\alpha_0 \leq \beta_1$. Ще докажем тези връзки между мерките α_0, α_1, β_0, β_1.

▷ *ТЕОРЕМА 5.15. Нека $G = (V, A)$ е граф и $S \subseteq V$. Множеството S е независимо множество от върхове в G тогава и само тогава, когато $V - S$ е върхово покритие.*

Доказателство: По дефиниция S е независимо множество в G тогава и само тогава, когато на нито едно ребро на G двата му края не са в S, т.е. тогава и само тогава, когато на всяко ребро в графа G поне единият му край е във $V - S$. От определението за върхово покритие следва верността на твърдението. ◁

▷ *ТЕОРЕМА 5.16. Ако $G = (V, A)$ е прост граф с n върха, то*

(5.16) $$\alpha_0 + \beta_0 = n.$$

Доказателство: Да означим със S_{max} и K_{min} съответно максималното независимо множество и минималното върхово покритие в G. Тогава

$$|S_{max}| = \alpha_0 \quad \text{и} \quad |K_{min}| = \beta_0.$$

Според теорема 5.15, $V - S_{max}$ е върхово покритие, а $V - K_{min}$ е независимо множество. Следователно

$$|V - S_{max}| = n - \alpha_0 \geq \beta_0, \tag{5.17}$$

$$|V - K_{min}| = n - \beta_0 \leq \alpha_0. \tag{5.18}$$

От (5.17) и (5.18) следва $\alpha_0 + \beta_0 \leq n$ и $\alpha_0 + \beta_0 \geq n$, откъдето следва (5.16). ◁

▷ *ТЕОРЕМА 5.17. Нека $G = (V, A)$ е граф без изолирани върхове. Тогава*

$$\alpha_1 \leq \beta_0, \tag{5.19}$$

$$\alpha_0 \leq \beta_1. \tag{5.20}$$

Доказателство: Нека M_{max} е произволно максимално сдвоване и K_{min} е произволно минимално върхово покритие. За покритието на ребрата от M_{max} са нужни поне $|M_{max}|$ върхове. Тогава очевидно всяко върхово покритие (в това число и минималното) трябва да съдържа поне $|M_{max}|$ върха, т.е. $|M_{max}| \leq |K_{min}|$, т.е.

$$\alpha_1 \leq \beta_0.$$

Нека сега S_{max} е максимално независимо множество от върхове в G и C_{min} е минимално ребрено покритие в G. За покриването на върховете от множеството S_{max} са необходими $|S_{max}|$ ребра. Следователно, всяко ребрено покритие трябва да съдържа поне $|S_{max}|$ ребра, т.е.

$$|S_{max}| \leq |C_{min}|,$$

откъдето

$$\alpha_0 \leq \beta_1.$$

С това доказателството е направено. ◁

В общия случай равенствата (5.19) и (5.20) не са изпълнени. Ще докажем обаче следната теорема:

▷ *ТЕОРЕМА 5.18. Нека $G = (X, Y, A)$ е биполярен граф без изолирани върхове. Тогава*

(5.21) $$\alpha_1 = \beta_0,$$

(5.22) $$\alpha_0 = \beta_1.$$

Доказателство: Нека M_{max} е максимално сдвояване и K_{min} е минимално върхово покритие. Да разгледаме произволно подмножество от върхове на X, т.е. $B \subseteq X$. Всяко ребро в G е инцидентно с връх от B или допълнението му $X - B$. От друга страна, всяко ребро, инцидентно с връх от B е инцидентно с връх от $\Gamma(B)$. Следователно $(X - B) \cup \Gamma(B)$ се явява върхово покритие в графа G. Тогава

$$|(X - B)| + |\Gamma(B)| \geq |K_{min}| = \beta_0.$$

От теоремата на Хол (вж. теорема 5.7)

$$|M_{max}| = \min_{B \subseteq X}\{|(X - B)| + |\Gamma(B)|\} \geq |K_{min}|, \quad \text{т.е.} \quad \alpha_1 \geq \beta_0.$$

От последното неравенство и (5.19) следва (5.21)

$$\alpha_1 = \beta_0.$$

Сега ще докажем (5.22). От теорема 5.16, $\alpha_0 + \beta_0 = n$, т.е.

(5.23) $$\alpha_0 = n - \beta_0.$$

От друга страна от теорема 5.14, $\alpha_1 + \beta_1 = n$, т.е.

$$\beta_1 = n - \alpha_1.$$

От (5.21), което вече доказахме, $\alpha_1 = \beta_0$, следователно

(5.24) $$\beta_1 = n - \beta_0.$$

От (5.23) и (5.24) следва (5.22), т.е.

$$\alpha_0 = \beta_1.$$

◁

Доказаната теорема 5.18 дава еквивалентни формулировки на теоремата на Хол.

За графа $G = (V, \Gamma)$, минималното върхово покритие или още доминиращото множество от върхове K се определя като минимално множество от върхове $K \subseteq V$ такова, че за всеки връх $v_j \notin K$ съществува дъга, входяща във v_j с начало връх от K, т.е. $K \cup \Gamma(K) = V$.

С езика на матрично представяне на графи задачата за намиране на минимално доминиращо множество от върхове се формулира така: Ако B^t е транспонираната матрица на съседство с единични диагонални елементи, да се намери такова минимално множество от стълбове в B^t, за което всеки ред на матрицата съдържа 1-ца в поне един от избраните стълбове.

Най-общо задачата за намиране на минимално множество от стълбове, "покриващо" всички редове на една $(0, 1)$-матрица, се нарича задача за минимално покритие (ЗМП). В тази задача матрицата не е задължително да бъде квадратна (както в случая на доминиращо върхово покритие) и освен това, на всеки стълб j в матрицата обикновено се съпоставя тегло c_j, като се търси покритие с минимално сумарно тегло.

Ясно е, че задачата за намиране на минимално върхово доминиращо множество е частен случай на ЗМП при $c_j = 1$, за $\forall j = 1,\ 2,\ \ldots,\ n$.

Теоритико-множествената интерпретация на ЗМП се състои в следното.

Дадено е множество $R = \{r_1,\ r_2,\ \ldots,\ r_m\}$ и фамилия F,

$$F = \{F_1,\ F_2,\ \ldots,\ F_n\}$$

от подмножества на R, т.е. $F_j \subset R$. Всяка подфамилия $F' \subset F$,

$$F' = \{F_{j_1},\ F_{j_2},\ \ldots,\ F_{j_k}\},$$

за която

$$\bigcup_{i=1}^{k} F_{ji} = R$$

се нарича *покритие на R*, а множествата F_{ji} се наричат *покриващи множества*.

Ако в допълнение поискаме покриващите множества да са непресичащи се две по две, покритието се нарича още *разбиване на R*.

Ако на всяко множество F_j е съпоставено (положително) тегло c_j, то ЗМП се формулира като задача за намиране покритие на R, имащо минимално сумарно тегло (теглото на F' се определя като $\sum c_{ji}$, $i = 1, ..., k$).

Аналогично се формулира и задачата за минимално разбиване (ЗМР).

В матрична форма ЗМП на матрицата $[t_{ij}]_{m\times n}$ може да се формулира като задача на линейното оптимиране по следния начин (вж. параграф 2.1, задача на линейното оптимиране):

$$\min L = \sum_{j=1}^{n} c_j x_j, \quad c_j \geq 0$$

при ограничения

$$\sum_{j=1}^{n} t_{ij} x_j \geq 1, \quad i = 1,\ 2,\ ...,\ m, \quad \text{където} \tag{5.25}$$

$$x_j = \begin{cases} 1, & \text{ако } F_j \in F', \\ 0, & \text{ако } F_j \notin F', \end{cases}$$

$$t_{ij} = \begin{cases} 1, & \text{ако } r_i \in F_j, \\ 0, & \text{ако } r_i \notin F_j. \end{cases}$$

В задачата за минимално разбиване (ЗМР) неравенствата (5.25) се обръщат в равенства, т.е. $\sum_{j=1}^{n} t_{ij} x_j = 1$, при $i = 1,\ 2,\ ...,\ m$.

ЗАДАЧА 5.14. [2] Трябва да се осигурят преводачи от 7 чужди езика на български език. Своите услуги предлагат 5 кандидата, чийто възможности (езици, които владее) са описани в таблица по следния начин (на всеки от преводачите заплащането е еднакво):

	A	B	C	D	E
език 1	1	0	1	1	0
език 2	1	1	0	0	0
език 3	0	1	0	0	0
език 4	1	0	0	1	0
език 5	0	0	1	0	0
език 6	0	1	1	0	1
език 7	0	0	0	1	1

Кои от 5-те кандидати да се подберат, така че да бъде осигурен превод с възможно най-малкия брой преводачи (минимално заплащане).

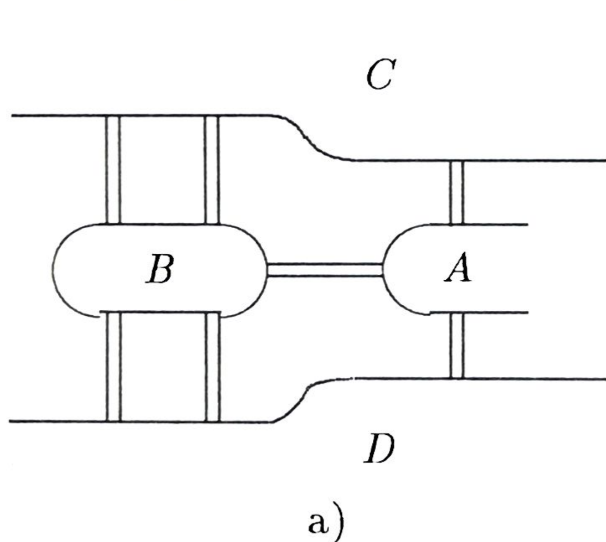
Отг.: B, C, D.

ЗАДАЧА 5.15. Задайте произволни тегла а всеки от стълбовете в матрицата от задача 5.14, т.е. задайте съответни заплащания на всеки от кандидатите в предишната задача. Определете покритието с минимално тегло.

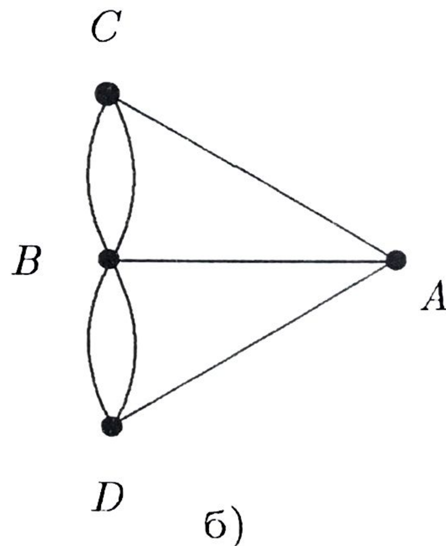
1.6. Ойлерови и Хамилтонови графи

1. Ойлерови графи

Да се върнем отново към задачата за Кьонигсбергските мостове, формулирана в началото на тази глава.



а)



б)

Ойлер пръв е поставил въпроса за съществуването на цикъл в неориентиран s-граф G, който преминава по всяко ребро точно веднъж. Всеки такъв цикъл (затворена верига, затворен път, затворен маршрут) се нарича *ойлеров цикъл*.

Когато съществува ойлеров цикъл, който покрива всички ребра на G, графът G се нарича *ойлеров граф*.

Отворена верига (път, маршрут), която не минава два пъти по едно и също ребро, се нарича *отворена ойлерова верига*.

На езика на графите задачата за мостовете лесно се превежда така (виж горе черт. б)). Ако на всеки участък от сушата съпоставим връх на граф, в който дъгите са съществуващите Кьонигсбергски мостове, поставеният въпрос "може ли жител

да излезе от дома си и да се върне пак там, като мине по всички мостове точно веднъж (ни повече, ни по-малко)" е всъщност въпрос съществува ли в графа от черт. б) ойлеров цикъл.

В конкретната ситуация е почти очевидно, че ойлеров цикъл не съществува, тъй като всички върхове на графа от черт. б) са от нечетна степен — броят на "излизанията" от всеки начален връх е по-голям от броя на "влизанията". В общия случай е вярна следната теорема:

▷ *ТЕОРЕМА 6.1. Свързаният неориентиран s-граф G съдържа покриващ ойлеров цикъл (е ойлеров) тогава и само тогава, когато всички върхове на графа са от четна степен.*

Доказателство: *Необходимост:* Очевидно, ако съществува покриващ ойлеров цикъл в графа G, то всички върхове на G са от четна степен (броят на "влизанията" в един връх е равен на броя на "излизанията").

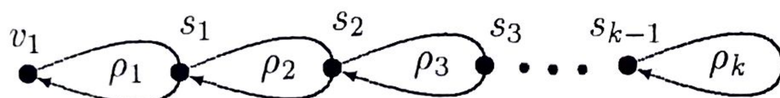
Достатъчност: Ще дадем конструктивно доказателство, което представлява алгоритъм за намиране на ойлеров цикъл в случая, когато всички върхове на графа са от четна степен.

Да си изберем произволен връх v_i в графа $G := G_i$ и да се "движим" произволно по различни ребра на графа. Рано или късно ще достигнем отново във върха v_i. Допускането, че се намираме някъде другаде (в друг връх) и не можем да се върнем във v_i, противоречи на предположението, че всички върхове на свързания граф са от четна степен. С други думи намерихме цикъл $\rho_i = (v_i, \ldots, v_i)$, включващ различни ребра на графа.

1 случай. Ако ребрата на ρ_i са всички ребра на G_i, то сме намерили ойлеров цикъл.

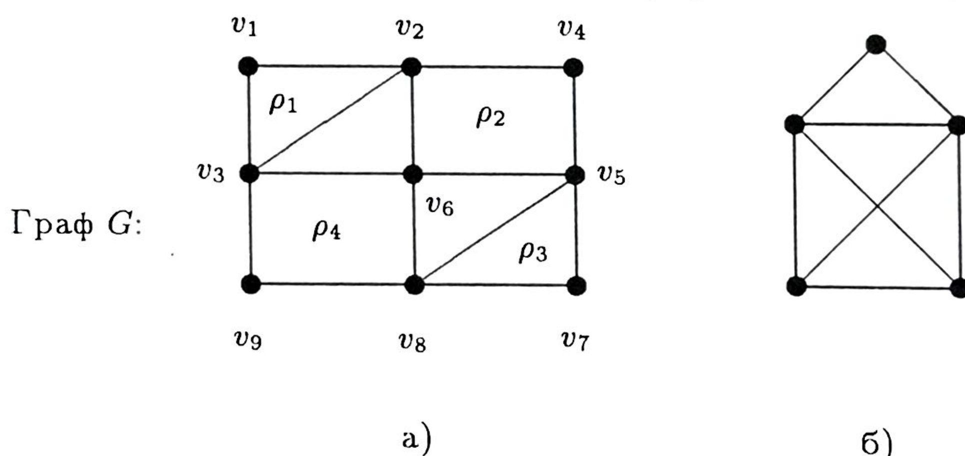
2 случай. Цикълът ρ_i не включва всички ребра на графа G_i. "Отстраняваме" (от разглеждане) ребрата на ρ_i от G_i. Избираме си връх s_i от цикъла ρ_i, инцидентен с някое от оставащите (неотстранени) ребра на G_i. Поне един такъв връх s_i (свързваща точка) със сигурност съществува (графът е свързан и всички върхове са от четна степен). Полагаме $v_i := s_i$ и повтаряме гореописаната процедура.

По-просто казано — "разбиваме" графа на k-ребрено непресичащи се цикли по следния начин:



Очевидно ойлеровия цикъл в графа G можем да опишем (начертаем без вдигане на молива и повтаряне на ребра) по следния начин. Тръгваме от v_1 и се движим в цикъла ρ_1, докато стигнем до свързваща точка s_1 (посоката на движение няма значение); от s_1 се движим в цикъла ρ_2, докато достигнем до свързващата точка s_2 и т.н., докато достигнем свързващата точка s_{k-1} и опишем изцяло цикъла ρ_k (последният цикъл); връщаме се в предпоследния цикъл и се движим по неописания му участък, докато стигнем до свързваща точка s_{k-2} и т.н., докато стигнем до свързваща точка s_1 и опишем неизвървяния участък от цикъла ρ_1, докато стигнем във v_1. ◁

ПРИМЕР 6.1. Има ли ойлеров цикъл в графа G и ако има, го задайте.



а) б)

а) Всички върхове на графа G са от четна степен и съгласно доказаната теорема графът е ойлеров, т.е. в него съществува покриващ ойлеров цикъл.

Разглеждаме $\rho_1 = (v_1,\ v_2,\ v_3,\ v_1)$. "Отстраняваме" ребрата от този цикъл и в оставащия граф разглеждаме цикълът $\rho_2 = (v_2,\ v_4,\ v_5,\ v_6,\ v_2)$. "Отстраняваме" и тези ребра от графа. Разглеждаме в оставащия граф цикълът $\rho_3 = (v_5,\ v_7,\ v_8,\ v_5)$. "Отстраняваме" и тези ребра от графа. В оставащия граф разглеждаме цикъла $\rho_4 = (v_8,\ v_9,\ v_3,\ v_6,\ v_8)$. "Отстраняваме" и тези ребра от графа — край на процедурата за разбиване на графа (ребрата от ρ_4 са последните останали ребра на графа). Един ойлеров цикъл в графа G е $(v_1,\ v_2,\ v_4,\ v_5,\ v_7,\ v_8,\ v_9,\ v_3,\ v_6,\ v_8,\ v_5,\ v_6,\ v_2,\ v_3,\ v_1)$.

Намерете и "друг" ойлеров цикъл.

б) Графът G от подточка б) не е ойлеров, т.е. в него не съществува покриващ ойлеров цикъл, тъй като има върхове от нечетна степен (два върха от степен 3).

В този граф има обаче покриваща отворена ойлерова верига, свързваща двата върха от нечетна степен. Намерете я!

Ясно е, че ако графът не е свързан, ойлеров цикъл покриващ графа не съществува, тъй като няма верига водеща от една негова компонента в друга.

Очевидно е и, че ако съществува отворена ойлерова верига, покриваща графа, тя свързва два върха на графа, които са от нечетна степен.

Фльори [22], е дал един доста прост алгоритъм за построяване на покриващ ойлеров цикъл (когато той съществува) в неориентиран граф, който лесно може да се приложи и за ориентирани графи: *"Стартирайте от връх v и всеки път отстранявайте преминатото ребро. Не преминавайте по ребро, ако отстраняването на това ребро води до двукомпонентно разбиване на графа (без да се броят изолираните върхове)".*

Следният резултат характеризира графите, които могат да бъдат покрити с отворени ойлерови вериги:

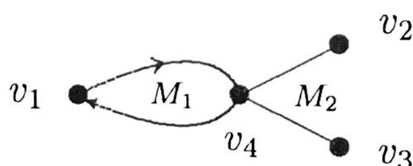
▹ *ТЕОРЕМА 6.2. Нека G е свързан граф с $2s > 0$ нечетни върха. Съществуват s независими (нямащи общи ребра) ойлерови вериги, които покриват G. По-малко от s вериги с това свойство няма. Във всяка система от s независими ойлерови вериги, покриващи G, веригите са отворени, две по две нямат общи крайни точки и всяка верига съединява върхове от нечетна степен* [13].

Доказателство: Нека t е минималният брой независими ойлерови вериги, които покриват G и нека

$$M = \{M_1,\ M_2,\ \dots,\ M_t\}$$

е едно минимално покритие на G.

1. Никое M_i, $i = 1,\ 2,\ \dots,\ t$, не е цикъл. Да допуснем, че например M_1 е цикъл. Оттук следва, че M_1 не може да има общ връх с никой от останалите вериги M_i. Наистина, ако допуснем, че цикълът M_1 има общ връх с веригата M_2 (отворена или затворена), т.е. графически имаме следната ситуация



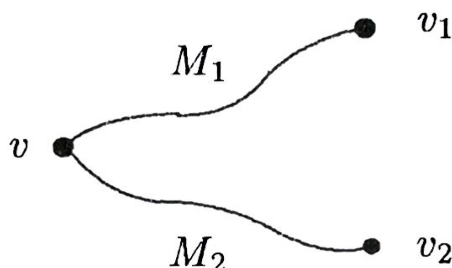
очевидно веригата $(v_2,\ v_4,\ v_1,\ v_4,\ v_3)$ е също ойлерова и ако заменим M_1 и M_2 с тази верига, ще получим покритие на G с $t-1$ на брой вериги, което противоречи на минималността на t.

И така, ако допуснем, че M_1 (произволна верига) е цикъл, следва, че този цикъл няма общ връх с останалите вериги M_i. Но това означава, че няма ребра, на които единият връх да е в M_1, а другият в някоя от останалите вериги. Тъй като графът

G е свързан и M е покритие на G, следва, че всички ребра на графа трябва да се включват в M_1, т.е. в графа съществува покриващ ойлеров цикъл, което е невъзможно, поради наличието на нечетни върхове ($2s > 0$) в графа и теорема 6.1.

И така, $\forall i$, M_i е отоврена верига.

2. Две по две веригите M_i, $i = 1, 2, \ldots, t$ нямат обща крайна точка. Ако допуснем противното, например M_1 и M_2 имат обща крайна точка, както е показано на чертежа долу



е ясно, че обединяването на двете вериги M_1 и M_2, и замяната им с веригата (v_1, v, v_2), която също е ойлерова, ще води до покритие на G с $t-1$ на брой маршрута, което противоречи на минималността на t.

3. Всяка верига съединява нечетни върхове. Верността на твърдението лесно следва от факта, че за началния край на веригата "излизанията" са с 1 повече от "влизанията", а за крайния връх на веригата — обратно. Оттук следва, че $2t = 2s$ или $s = t$. ◁

СЛЕДСТВИЕ 1. При $s = 1$, т.е. ако в графа G съществуват точно два върха от нечетна степен, графът G се покрива от една отворена ойлерова верига (вж. б) от пример 6.1).

ЗАДАЧА 6.1. Да се докаже, че шахматен кон, движейки се по шахматната дъска, не може да направи всеки допустим ход точно по веднъж.

Решение: Броят възможни ходове на коня от всяко шахматно поле е следният:

2	3	4	4	4	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

Ако разгледаме графа G с върхове 64-те шахматни полета и ребра, свързващи онези върхове (шахматни полета), между които има възможен ход на коня, е ясно, че всяка последователност от ходове може да се разглежда като верига в графа G и обратно. Въпросът, поставен в задачата е може ли графът G да се покрие с ойлерова верига.

В горната таблица числата са степените на върховете в графа G, т.е. броя на възможните ходове от това шахматно поле. Тъй като съществуват 8 върха с нечетна степен 3, съгласно доказаната теорема 6.2, минималният брой независими ойлерови вериги, покриващи G, е 4, т.е. не може с една ойлерова верига да се покрие графът G, което решава задачата.

ЗАДАЧА 6.2. Пресметнете броя на ребрата в графа G от задача 6.1.

Отг.: $m = \frac{1}{2}.(4.2 + 8.3 + 16.4 + 16.6 + 16.8) = 160.$

Покриването на един граф с ойлерови цикли или вериги е една важна за практиката задача. Ще формулираме няколко оптимизационни проблема, тясно свързани с намирането на ойлерови вериги и цикли.

(A) **Задача за китайския пощальон — The Chinese Postman Problem (CPP)**. Проблемът е формулиран и изследван от китайския математик Kwan Mei - Ko, откъдето идва и наименованието му. Пощальон трябва да разнесе писма, телеграми, пенсии и т.н. до всички адреси от региона, който обслужва. Какъв

маршрут трябва да избере пощальонът, за да измине минимално разстояние и да се върне отново в пощенския офис?

Ясно е, че обслужваният от пощальона регион може да се разглежда като граф, в който върховете са кръстовищата, а ребрата са улиците, по които преминава пощальона. Очевидно, ако графът е ойлеров, оптималното решение на проблема е пощальонът да си избере един от съществуващите ойлерови маршрути. Ако обаче регионът, който пощальонът обслужва е такъв, че съответният му граф не е ойлеров, то тогава очевидно пощальонът ще трябва да преминава повторно по някои от улиците. Как да бъде подбран маршрутът на движение, така че връщайки се в изходно положение (в пощенския офис), пощальонът да е изминал минимално сумарно разстояние?

Горният проблем е интересен класически оптимизационен проблем, тъй като може да се отнася не само до пощальони, а до движение на превозни средства или хора, доставящи стоки, извършващи инспекции на обекти, събиране на отпадъци и т.н и т.н. В следващата глава отново ще се върнем на подобен род проблеми и ще дадем алгоритми за тяхното решаване.

(Б) **Задача за снегорина**. Служба за поддръжка на пътищата при зимни условия е дислоцирана на определено място в даден град. Всяка сутрин кола (коли) на службата излиза от депото, за да почиства от сняг улиците и да разпръсква солена луга против заледяването. Известен е капацитетът на колата — например 5 тона солена луга. Да се намери маршрут за движение на снегорина (снегорините) при неколкократно презареждане с луга в депото, така че да бъдат обезопасени всички улици с минимални разходи. С други думи отново трябва да се реши проблем за покриване на граф, като условието е по всяка от улиците да се преминава поне по веднъж.

Очевидно могат да се формулират редица практически проблеми, свързани с покриване на графи.

Ще формулираме аналог на теорема 6.1, отнасящ се до ориентирани s-графи.

▷ ТЕОРЕМА 6.3. *Ориентираният свързан s-граф G съдържа покриващ ойлеров цикъл (ойлерова верига) тогава и само тогава, когато за всички полустепени на входа $d^-(v_i)$ и полустепени на изхода $d^+(v_i)$ са изпълнени условията:*

а) за цикъл — $\forall i \in V,\ d^-(v_i) = d^+(v_i)$;

б) за верига — $\forall i \neq s$ *и* t, $d^-(v_i) = d^+(v_i)$;

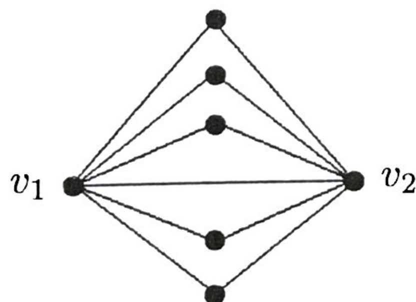
$$d^-(s) = d^+(s) - 1,$$
$$d^-(t) = d^+(t) + 1,$$

където s *и* t *са началният и крайният връх на ойлеровата верига.*
◁

От казаното дотук можем в резюме да заключим, че

	"графът G е ойлеров" $\Longleftrightarrow$ "степените на всички върхове са четни" $\Longleftrightarrow$ "графът G е обединение на ребрено непресичащи се цикли".

Ясно е, че всеки връх в ойлеров граф G се съдържа в някой цикъл. Може да се докаже, че когато всеки цикъл в ойлеровия граф G съдържа върха v, то тогава стартирайки от върха v по какъвто и начин да се движим произволно по ребрата на G, ще получим покриващ ойлеров цикъл. Такива ойлерови графи G се наричат *произволно-ойлерови относно върха* v. Да илюстрираме казаното със следния пример:



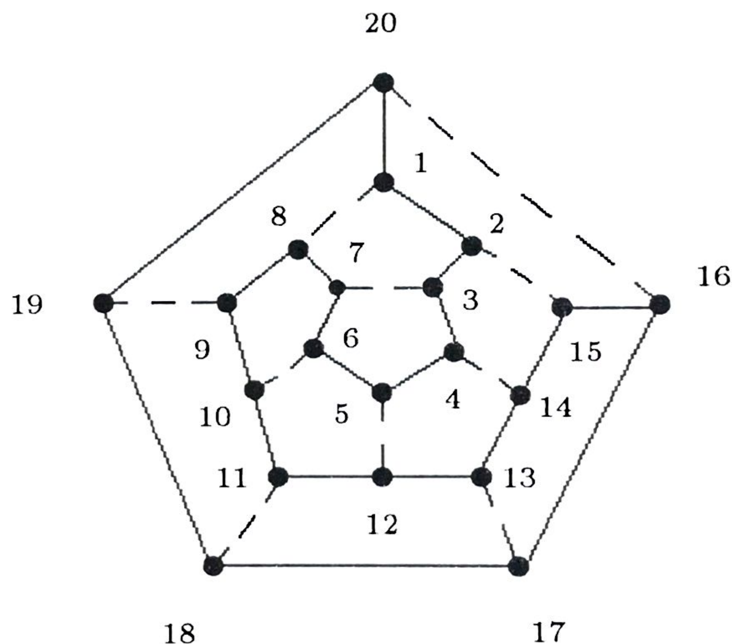
В изобразения граф върховете v_1 и v_2 принадлежат на всеки цикъл в графа, т.е. графът G е произволно-ойлеров относно v_1 и v_2. Следователно, тръгвайки от v_i, $i = 1, 2$ и движейки се напълно произволно по ребрата на графа (без да ги повтаряме), ще получим покриващ ойлеров цикъл. Останалите върхове на графа не притежават това свойство, т.е. графът не е произволно-ойлеров относно върховете $v_i \neq v_1, v_2$.

Графът G се нарича *произволно-ойлеров*, ако той е такъв относно всички свои върхове. Последният изобразен граф не е произволно-ойлеров.

2. Хамилтонови графи

Един граф G се нарича *хамилтонов граф*, ако съществува прост цикъл, минаващ през всички върхове на графа G. Такъв цикъл се нарича *хамилтонов цикъл*.

През 1859 г. сър Уилям Хамилтон е поставил пръв проблема дали за графа, образуван от върховете и ребрата на додекаедъра (изпъкнал многостен с 12 стени) съществува прост цикъл, който минава през всички върхове. Равнинната реализация на графа е показана на чертежа долу, като е посочен и един хамилтонов цикъл:



Хамилтонова верига (път, маршрут) в G — това е проста верига, съдържаща всички върхове на графа G.

ЗАДАЧА 6.3. Да се покаже, че е възможно шахматен кон, тръгвайки от някакво шахматно поле да премине по всяко друго шахматно поле точно веднъж и да се върне в изходно положение.

Читателят лесно може сам да намери едно от многото решения на тази задача. По-долу е дадено едно възможно решение

36	53	38	57	34	29	40	59
13	56	35	52	39	58	33	28
54	37	12	43	30	51	60	41
11	14	55	50	21	42	27	32
6	49	10	23	44	31	20	61
15	2	5	48	9	22	45	26
4	7	64	17	24	47	62	19
1	16	3	8	63	18	25	46

на задачата, като числата във всяко от шахматните полета описват последователните позиции на коня.

Съществуват два основни класа задачи, свързани с хамилтоновите цикли.

(В) Ако G е ориентиран граф, да се намери хамилтонов цикъл (или всички цикли), ако съществува.

(Г) Даден е пълен ориентиран граф G. На всяка от дъгите на G е съпоставено тегло c_{ij} (ако G не е пълен, можем да го разглеждаме като пълен ориентиран граф, като съпоставим на отсъстващите дъги тегло равно на ∞). Да се намери хамилтонов цикъл (верига), на който сумарното тегло е минимално.

Тази задача е известна в литературата като *задача за търговския пътник* — The Traveling Salesman Problem (TSP).

Очевидно задача (В) е частен случай на задача (Г). Наистина, ако в задача (В) дадем произволни крайни тегла на дъгите, получаваме задача (Г). Ако решението на последната задача е крайно, т.е. хамилтоновият цикъл с минимално тегло има тегло различно от ∞, то това решение се явява хамилтонов цикъл за графа (В), т.е. решение на задача (В). В противен случай, т.е. когато решението на (Г) е ∞, то задача (В) няма решение.

Формулираната (TSP)-задача очевидно има *минисумарен характер.* На други места в литературата се разглежда една *минимаксна модификация* на (TSP). В нея се търси такъв хамилтонов цикъл, в който най-дългата дъга е минимална, т.е. дъгата с най-голямо тегло е минимална. Може да се покаже, че задача (В) и минимаксната (TSP) задача са еквивалентни, т.е. алгоритъмът за намиране на хамилтонов цикъл в ориентиран граф, решава и минимаксната (TSP) задача и обратно.

В следващата глава ще видим, че редица практически проблеми се интерпретират като минисумарни или минимаксни задачи.

Въпреки, че задача (В) е частен случай от задача (Г), за решаването на задача (В) също са разработени методи, тъй като сама по себе си задача (В) се решава по-просто. В следващата глава ще разгледаме алгоритми за решаването на (TSP).

Очевидно е, че ако графът G е хамилтонов, то той е свързан. Освен това, ако в графа G съществуват два върха s и t, за които съществува единствен прост $(s-t)$ път, то очевидно графът G не е хамилтонов. Интуицията подсказва, че в графи с "достатъчно много" ребра има хамилтонов цикъл. Например във всеки пълен граф $(n \geq 3)$ има хамилтонов цикъл. Нещо повече, в пълния граф има $\frac{(n-1)!}{2}$ хамилтонови цикли. Решаването обаче на задачата за търговския пътник "с мускули", т.е. пораждането на всички хамилтонови цикли и избирането от тях на този с минимално (максимално) тегло, не е приложимо на практика в реално време.

За съжаление, за хамилтоновте графи не съществува сравнително проста и елегантна характеризация, каквато за ойлеровите графи съществува. Известни са няколко достатъчни условия простият граф G да бъде хамилтонов, без обаче тези условия да се явяват необходими.

Например, ако за всеки два несъседни върха s и t в G е изпълнено

$$d(s) + d(t) \geq n,$$

то G е хамилтонов граф. Това условие обаче не е необходимо.

Графът на движение на шахматното конче от задача 6.3 е хамилтонов, като за него за всяко s и t, $d(s) + d(t) \leq 16$.

Ще формулираме няколко условия за хамилтоновост.

▷ *ТЕОРЕМА 6.4. Нека $G = (V, A)$ е прост граф $(n \geq 3)$ със сте-*

пени на върховете $d(v_1) \le d(v_2) \le \ldots \le d(v_n)$. *Ако*

$$(6.1) \qquad d(v_k) \le k < \frac{1}{2}n \implies d(v_{n-k}) \ge n - k,$$

то графът G *е хамилтонов.* ◁

СЛЕДСТВИЕ 1. Ако $1 \le k \le n \implies d(v_k) \ge \frac{1}{2}n$, *то* G *е хамилтонов.*

СЛЕДСТВИЕ 2. Ако $(s,t) \notin A \implies d(s) + d(t) \ge n$, *то* G *е хамилтонов.*

СЛЕДСТВИЕ 3. Ако $1 \le k \le \frac{1}{2}n \implies d(v_k) > k$, *то* G *е хамилтонов.*

СЛЕДСТВИЕ 4. Ако $j < k$, $d(v_j) \le j$, $d(v_k) \le k-1 \implies d(v_j) + d(v_k) \ge n$, *то* G *е хамилтонов.*

Доказателството на теоремата можете да намерите в [3] или да направите самостоятелно като упражнение, а следствията се доказват като се използва това, че условията в тях влекат (6.1) [23]—[26].

Ойлеровите и хамилтоновите цикли имат в известен смисъл двойнствен характер (замяна на връх с ребро и обратно) — при едните се преминава през всяко ребро веднъж, а при другите през всеки връх веднъж.

Ако G е граф, то *ребрен граф* $L(G)$, съответен на графа G се определя по следния начин: $L(G)$ има толкова върхове, колкото са ребрата в G и два върха u и v на $L(G)$ са съединени с ребро тогава и само тогава, когато ребрата от G съответни на u и v, са съседни ребра в G.

Лесно може да се покаже [27], че:

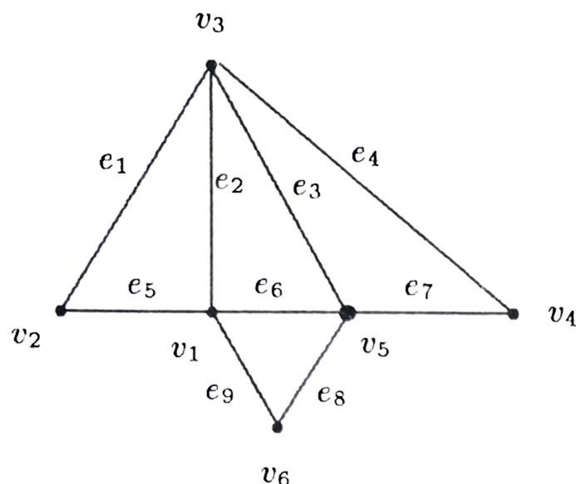
а) ако графът G е ойлеров, неговият ребрен граф $L(G)$ също е ойлеров;

б) ако графът G е ойлеров, то $L(G)$ е хамилтонов;

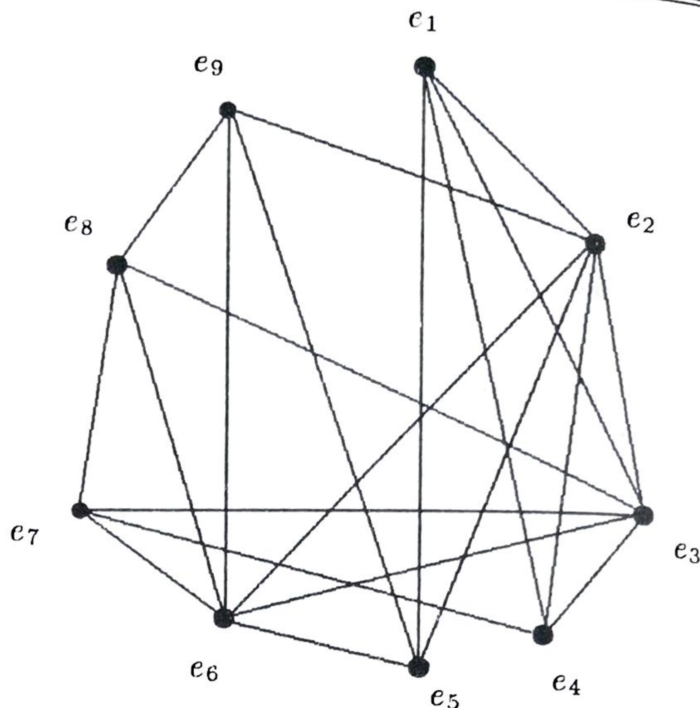
в) ако графът G е хамилтонов, то $L(G)$ също е хамилтонов.

Обратните на твърденията а), б) и в) не са верни, което лесно се показва с контрапримери.

ПРИМЕР 6.2. На чертежа долу е изобразен граф G и съответният му ребрен граф $L(G)$:



Граф G



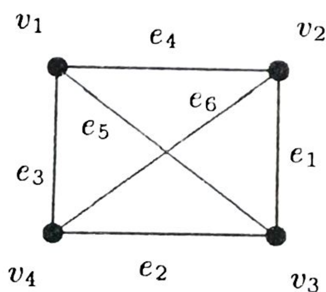
Граф $L(G)$

Очевидно за графа G има покриващ ойлеров цикъл, тъй като всичките му върхове са от четна степен, т.е. G е ойлеров. Освен това графът G е хамилтонов, например $(v_1, v_2, v_3, v_4, v_5, v_6, v_1)$ е хамилтонов цикъл. Графът $L(G)$ също е ойлеров, тъй като степените на неговите върхове отново са четни. В графа $L(G)$ съществува и хамилтонов цикъл, зададен с върховете $(e_3, e_1, e_4, e_7, e_6, e_8, e_9, e_5, e_2, e_3)$.

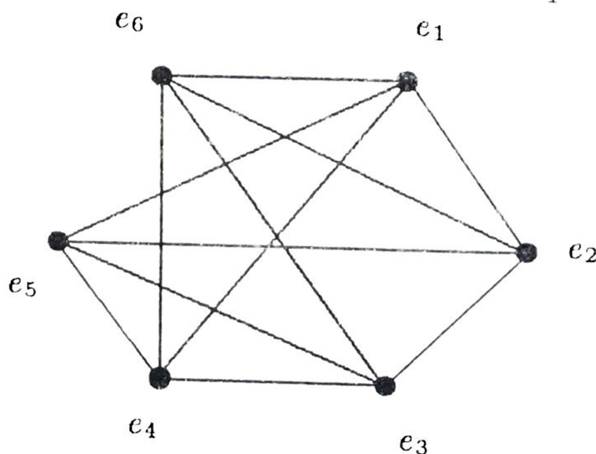
С други думи, пример 6.2 илюстрира а), б) и в).

Обратното на твърдение а) не е вярно, което се вижда от следния

ПРИМЕР 6.3. На чертежа долу $L(G)$ е ойлеров, а G не е ойлеров.



Граф G



Граф $L(G)$

В пример 6.2, ако премахнем реброто e_6 в графа G, ще получим граф G', който не е ойлеров. Но $L(G')$ е хамилтонов ($L(G')$ е

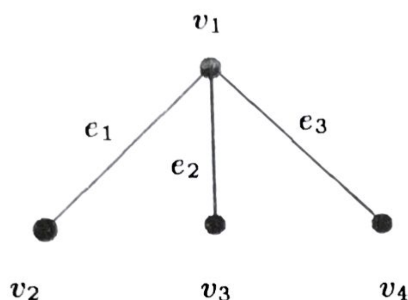
същият като $L(G)$, но в него няма връх e_6 и инцидентните с него ребра). В $L(G')$ един хамилтонов цикъл е последователността от върхове

$$(e_3,\ e_1,\ e_4,\ e_7,\ e_8,\ e_9,\ e_5,\ e_2,\ e_3),$$

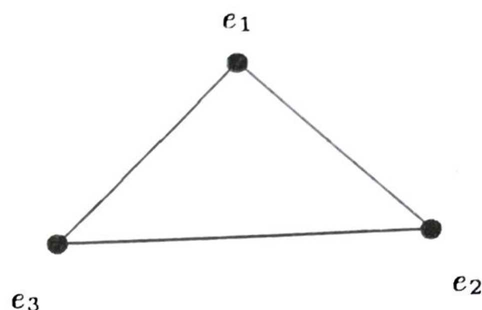
т.е. обратното на твърдение б) не е вярно.

Ще покажем, че обратното на твърдение в) също не е вярно.

ПРИМЕР 6.4. Да разгледаме



Граф G



Граф $L(G)$

Очевидно $L(G)$ е хамилтонов, а G не е хамилтонов граф.

В края на този параграф ще дадем едно по-формално описание на алгоритъм за търсене на ойлеров цикъл в граф $G = (V, A)$, дадено в [17]. Използвана е неформална версия на езика Паскал, която дава възможност идеята на алгоритъма да не се размива. Освен това ще дадем и един алгебричен метод, реализиращ алгоритъм за намиране на всички хамилтонови вериги и цикли в ориентиран граф.

АЛГОРИТЪМ ЗА ТЪРСЕНЕ НА ОЙЛЕРОВ ЦИКЪЛ.

Данни: Свързан граф $G = (V, A)$ с четни степени на върховете, представен със списъци $ЗАПИС\ [v]$, $v \in V$.

Резултати: Ойлеров цикъл, представен с последователност от върхове в стека *СТЕК 2*.

```
begin
    СТЕК 1 := ∅;  СТЕК 2 := ∅;
    v := произволен връх на графа;
    СТЕК 1 ⟸ v;
    while СТЕК 1 ≠ ∅ do
        begin v := top (СТЕК 1);
```

```
        (*v = елемента от върха на стека*)
7           if ЗАПИС [v] ≠ ∅ then
8              begin u := първия връх на списъка ЗАПИС [v];
9                 СТЕК 1 ⟸ u;
                  (* отстраняване на ребро {v, u} от G*)
10                ЗАПИС [v] := ЗАПИС [v] \ {u};
                  ЗАПИС [u] := ЗАПИС [u] \ {v};
11                v := u
12             end
13          else (* ЗАПИС [v] = ∅ *)
14             begin v ⟸ СТЕК 1; СТЕК 2 ⟸ v
15             end
16       end
17 end
```

Цикълът от ред 5 започва да строи път с начало върха v_0 (избран в ред 3), като върховете на пътя се поставят в *СТЕК 1*, а ребрата се отстраняват от графа. Тези действия продължават, докато в ред 7 *ЗАПИС* $[v] = \emptyset$. Тогава всъщност $v = v_0$, тъй като всички върхове са четни. По този начин от графа се отстранява цикъл, а върховете на този цикъл се намират в *СТЕК 1*. Както отбелязахме по-рано в този параграф, в получения (модифицирания) граф степените на върховете продължават да бъдат четни. Върхът $v = v_0$ се прехвърля от *СТЕК 1* в *СТЕК 2*, а поредният връх v става връх в стека *СТЕК 1*. Процесът се повтаря от този връх (ако *ЗАПИС* $[v] \neq \emptyset$) като се намира и се поставя в *СТЕК 1* цикъл, минаващ през върха v. Това продължава, докато *СТЕК 1* не стане празен. Ясно е, че върховете в *СТЕК 2* образуват път, тъй като върхът v се пренася в *СТЕК 2* точно когато *ЗАПИС* $[v] = \emptyset$, т.е. когато всички инцидентни с v ребра са представени (като двойки съседни върхове) в един от стековете. Следователно, след изпълнение на алгоритъма, *СТЕК 2* съдържа ойлеров цикъл.

Сложността на разгледания алгоритъм е $O(m)$. За сложност на алгоритми виж началото на параграф 1.8 и параграф 2.4.

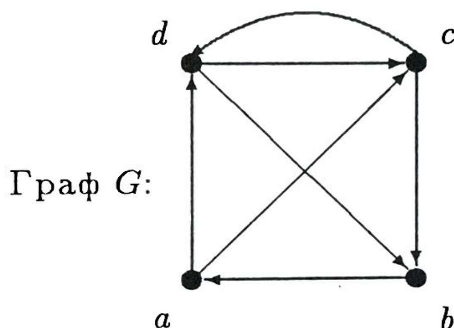
В параграф 1.4 "Матрично представяне на графи", дефинирахме понятието матрица на съседство. Ще дадем алгоритъм за намиране на всички хамилтонови вериги в ориентиран граф

$G = (V, E)$, използващ матрицата на съседство и правилата за умножение на матрици. Алгоритъмът се основава на резултатите, получени в [51] - [53].

За целта ще дефинираме понятието "*вътрешно произведение на върхове*" за пътя $x_1, x_2, \ldots, x_{k-1}, x_k$, което се определя като израз от вида $x_2 x_3 \ldots x_{k-1}$, несъдържащ крайните върхове x_1 и x_k (при $k = 2$, произведението е равно на 1).

Ако B е матрица на съседство, то под *модифицирана матрица на съседство* B_{mod} се разбира матрицата с n реда и n стълба и елементи $\beta(i,j) = x_j$, ако съществува дъга от x_i до x_j и $\beta(i,j) = 0$, в противен случай.

ПРИМЕР 6.5. Да разгледаме следния граф:



Неговите матрица на съседство и модифицирана матрица на съседство са следните:

$$B = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}; \quad B_{mod} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & c & d \\ a & 0 & 0 & 0 \\ 0 & b & 0 & d \\ 0 & b & c & 0 \end{pmatrix} \end{array}.$$

Ако ние имаме матрицата $P_l = [p_l(i,j)]$, чиито елементи са сума от вътрешните произведения на всички прости пътища с дължина $l \geq 1$ между различните върхове x_i и x_j, можем да направим следното. Полагаме $p_l(i,i) = 0$ за $\forall i$ и разглеждаме произведението на матриците $B_{mod}.P_l$, получено с правилото за умножение ред по стълб, т.е.

$$B_{mod}.P_l = P'_{l+1}[p'_{l+1}(s,t)],$$

където

$$p'_{l+1}(s,t) = \sum_k \beta(s,k).p_l(k,t). \qquad (6.2)$$

Очевидно $p'_{l+1}(s,t)$ се явява сума от вътрешните произведения на всички пътища от x_s до x_t с дължина $l+1$. Тъй като пътищата от x_k до x_t, представени с вътрешните произведения от $p_l(k,t)$, са прости, то измежду пътищата, получавани с помощта на (6.2), може да се появят непрости пътища. Такива са само тези пътища, вътрешните произведения на които в $p_l(k,t)$ съдържат върха x_s. Следователно, след проста проверка и отстраняване от $p'_{l+1}(s,t)$ на всички събираеми, съдържащи x_s, ще получим матрица $P_{l+1} = [p_{l+1}(s,t)]$, в която диагоналните елементи са 0 и задаваща простите пътища с дължина $l+1$.

Ще илюстрираме казаното с помощта на матриците от пример 6.5.

Ако положим $B \equiv P_1$ (прости пътища с дължина 1) и умножим матриците B_{mod} и P_1, ще получим $P'_2 = B_{mod}.P_1$, даваща пътищата между всеки два върха с дължина 2.

$$P'_2 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left(\begin{array}{cccc} 0 & c+d & d & c \\ 0 & 0 & a & a \\ b & d & \underline{d} & 0 \\ b & c & 0 & \underline{c} \end{array}\right) \end{array}; \qquad P_2 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left(\begin{array}{cccc} 0 & c+d & d & c \\ 0 & 0 & a & a \\ b & d & 0 & 0 \\ b & c & 0 & 0 \end{array}\right) \end{array}.$$

Матрицата P_2 дава простите пътища с дължина 2.

В общия случай, изчислявайки $B_{mod}.P_{l+1}$, ще намерим P_{l+2} и т.н., ще намерим матрицата P_{n-1}, даваща хамилтоновите пътища с дължина $n-1$ между всички двойки върхове. Очевидно хамилтоновите цикли ще се получават от пътищата в P_{n-1} и тези дъги на графа, които съединяват началния и крайния връх на всеки път. С други думи, хамилтоновите цикли се задават от $B.P_{n-1}$.

В качество на първоначална матрица P_1 се взима матрицата на съседство B, в която диагоналните елементи (ако се налага) се зануляват.

Ако продължим да прилагаме дадения алгоритъм за графа от пример 6.5, на следващата итерация ще получим $P'_3 = B_{mod}.P_2$ и P_3:

$$P'_3 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left(\begin{array}{cccc} \underline{cb+db} & cd+dc & 0 & 0 \\ 0 & \underline{ac+ad} & ad & ac \\ db & \underline{dc} & \underline{ba} & ba \\ cb & \underline{cd} & ba & \underline{ba} \end{array}\right) \end{array};$$

$$P_3 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{cccc} a & b & c & d \\ \hline \end{array}$$

$$P_3 = \begin{pmatrix} 0 & cd+dc & 0 & 0 \\ 0 & 0 & ad & ac \\ db & 0 & 0 & ba \\ cb & 0 & ba & 0 \end{pmatrix}.$$

Хамилтоновите пътища (вериги) съответни на елемента $(1, 2)$ от P_3 са $acdb$ и $adcb$ и те дават съответно хамилтоновите цикли $acdba$ и $adcba$, ако добавим затварящата дъга (b, a). Останалите хамилтонови вериги в P_3 водят до същите хамилтонови цикли — $badcb$, $bacdb$, $cdbac$, $cbadc$, $dcbad$, $dbacd$.

Очевидно предложеният алгоритъм е твърде неефективен. В процеса на нарастване на l, елементите на матрицата P_l ще включват все по-голям брой членове. Разбира се след достигане на някаква критична стойност за l, броят на членовете ще започне да намалява. Това следва от факта, че при малки стойности на l и големи размери на графа, броят пътища с дължина $l+1$ обикновено е по-голям от броя пътища с дължина l, а за големи значения на l ситуацията е обратна. С други думи обемът памет, необходим за съхраняване на матриците P_l расте твърде бързо, докато стигне до максимум за някоя критична стойност на l.

Ако се интересуваме само от хамилтоновите цикли, а от казаното е ясно, че те могат да се получат от членовете на вътрешното произведение на произволна диагонална клетка на матрицата $B_{mod}.P_{n-1}$, то е достатъчно да знаем само елемента $p_{n-1}(1, 1)$. Не е необходимо на всеки етап да се изчисляват и съхраняват напълно матриците P_l, достатъчно е да се намира първия стълб от P_l. Тази малка модификация намалява обема на необходимата памет и времето за изчисление n пъти.

По-късно в параграф 1.8, за да илюстрираме метода за търсене с връщане назад, ще разгледаме още един алгоритъм за намиране на хамилтонови цикли.

1.7. Планарни и двойнствени графи. Оцветявания

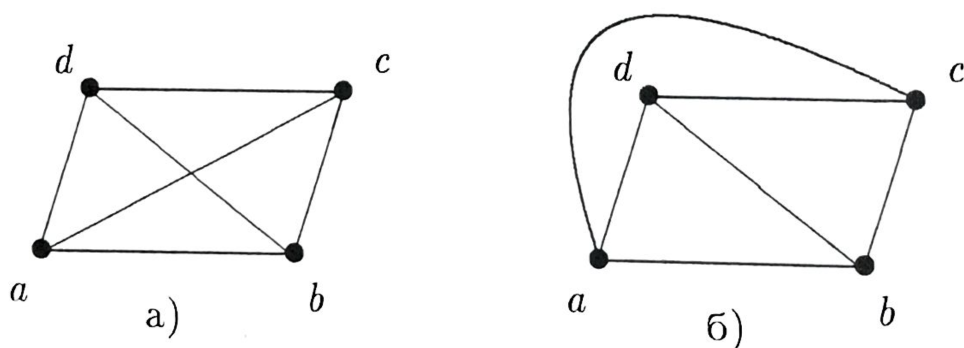
В този параграф ще разгледаме още няколко важни понятия и резултати от теория на графите. По-подробно ще се спрем на понятието планарност и ще дадем някои свойства на планарните графи. Не толкова подробно ще разгледаме двойнствеността, която е важна и интересна особено за физиката, където напрежението и токът в електрическите вериги са по същество

двойнствени променливи. Тази двойнственост следва от законите на Кирхоф и е свързана с двойнствеността на циклите и разрязващите множества.

1. Планарни графи

Най-общо *планарният граф* е граф, който може да се нарисува в равнината така, че никои две ребра на графа да не се пресичат (освен в крайните точки).

Графът на черт. 1.16 а) е планарен или още равнинен, тъй като както се вижда от подточка б) може да се изобрази така, че ребрата му да не се пресичат.

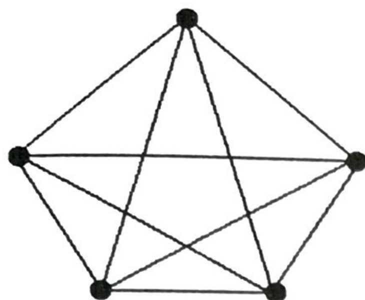


Черт. 1.16

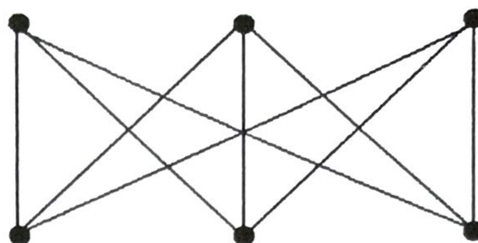
ЗАДАЧА 7.1. Изобразете планарния граф от черт. 1.16 б) така, че ребрата му да са отсечки.

Изучаването на планарните графи само на пръв поглед предполага топологични трудности. Това в действителност не е така.

Графите K_5 и $K_{3,3}$, изобразени по-долу, не са планарни и имат важно значение при характеризацията на планарността:



а) K_5



б) $K_{3,3}$

Черт. 1.17

Казваме, че *графът G се разполага* на повърхността S, ако може да бъде нарисуван на тази повърхност по такъв начин, че ребрата му да се пресичат само в крайните върхове. Споменатото изображение се наирча *планарно*.

Графът G е *планарен*, ако може да се разположи на равнина.

Ясно е, че ако в графа има примки или паралелни ребра, не съществува планарно изображение на този граф така, че всички негови ребра да са отсечки от прави линии. Може да се докаже обаче следното твърдение:

▷ *ТЕОРЕМА 7.1. За всеки прост планарен граф G съществува планарно изображение, при което ребрата са отсечки.* ◁

Интересно е следното твърдение:

▷ *ТЕОРЕМА 7.2. Графът G се разполага на равнина тогава и само тогава, когато може да се разположи на сфера.*

Доказателство: Да разгледаме сферата S и равнината α, които се допират. Да наречем условно допирната им точка южен полюс, а диаметрално противоположната точка N на сферата — северен полюс. Да разгледаме следното биективно изображение между сферата и равнината. "На произволна точка A от сферата се съпоставя точка A' от равнината и обратно, където A' е пресечната точка на правата NA с равнината α". Точката A' се нарича *стереографична проекция* на A върху равнината.

Нека G' е разполагане на графа G върху сферата S. Нека сферата S и равнината α се допират така, че северният полюс N на сферата не е нито връх на графа, нито точка от негово ребро при разполагането G'. Тогава очевидно стереографичната проекция на G' ще бъде разполагане на графа G върху

равнината α, тъй като всички ребра на G' се пресичат само в крайните си точки и изображението между точките на сферата и техните образи при стереографичното проектиране е биективно.

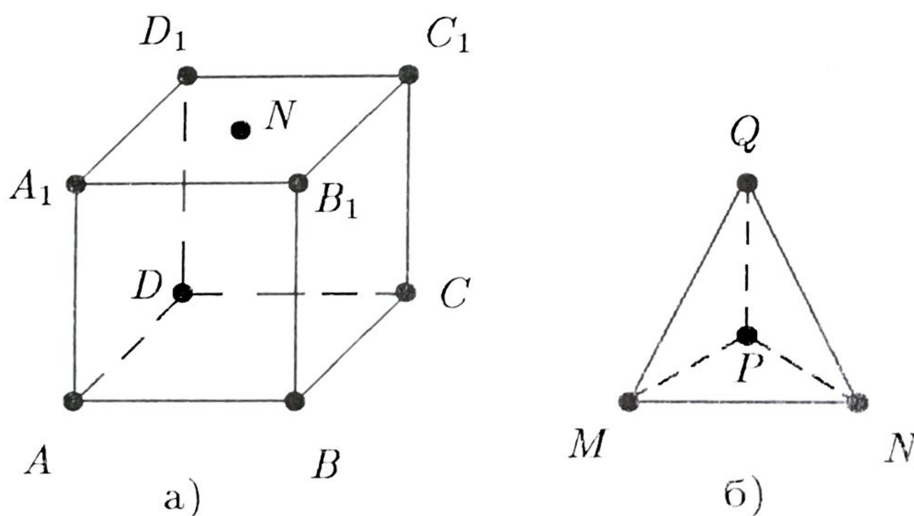
С аналогични разсъждения се установява, че ако G' е разполагане на графа G върху равнината α, то графът G се разполага и върху сферата S. ◁

Може да се докаже и следната теорема [29]:

▷ *ТЕОРЕМА 7.3. Разделимият граф G е планарен тогава и само тогава, когато са планарни неговите блокове.* ◁

2. Формула на Ойлер

Да разгледаме куба $ABCDA_1B_1C_1D_1$ и тетраедъра $MNPQ$ (черт. 1.18), които са изпъкнали многостени.



Черт. 1.18

За куба, който има 6 стени, 8 върха и 12 ръба, е налице съотношението: броят на стените + броя на върховете − броя на ръбовете е 2, т.е. $6 + 8 - 12 = 2$.

Аналогична зависимост е в сила и за тетраедъра, където отново имаме $4 + 4 - 6 = 2$.

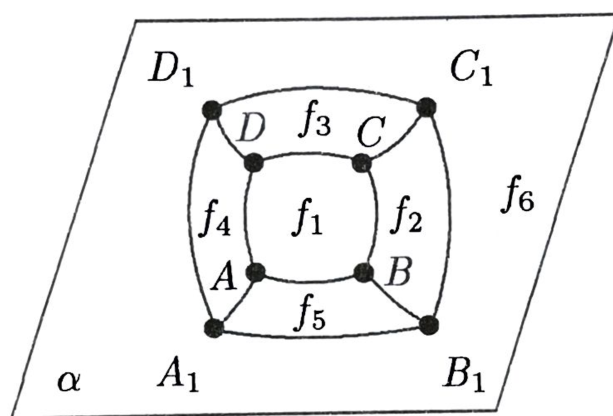
Тази зависимост между броя на върховете, стените и ръбовете на изпъкналите многостени е била известна още на Рене Декарт (1620 г.) и е доказана от Леонард Ойлер през 1758 г.

▷ *ТЕОРЕМА 7.4. (Теорема на Ойлер) За всеки изпъкнал многостен броят на върховете плюс броя на стените минус броя на ръбовете е 2.* ◁

Разполагането на всеки планарен граф върху равнина я разделя на *области* — тези с крайна площ се наричат *крайни области*, в противен случай — *безкрайни (неограничени) области.*

Очевидно всяка област при сферично разполагане на планарен граф е крайна.

Кубът или друг произволен изпъкнал многостен очевидно може да се разположи върху сфера S, която се допира до равнината α така, че северният полюс N на сферата да се намира "вътре" в една от стените на многостена (вж. черт. 1.18 а)). Тогава стереографичната проекция на сферичното разполагане G' в равнината α, за куба ще изглежда така:



Очевидно в горния граф, представящ куба в равнината, върховете съответстват на върховете на многостена, ребрата са съответни на ръбовете на многостена. В случая стената $A_1B_1C_1D_1$ на куба съответства на най-външната област от горния чертеж. Един равнинен граф заедно с областите, определени от него, се нарича *равнинна карта*. Всеки равнинен граф има точно една неограничена област.

Граница на област е множеството от ребра, които я ограждат. Всяко ребро от прост цикъл в графа влиза в границата на две области.

Областите на равнинната карта често се наричат *страни*, като две страни са *съседни*, ако техните граници имат общо ребро.

От казаното е ясно също, че планарният граф може да се разположи върху равнината по такъв начин, че всяка произволно избрана област да стане безкрайна.

Да означим с f_1, f_2, ..., f_k областите на един планарен граф, като f_k е неограничената област (при куба и означенията от последния чертеж, областта f_6 е безкрайна). Нека C_i е цикълът на границата на областта f_i, $1 \leq i \leq k$. Очевидно, сумата по mod 2 на всеки $t \geq 2$ такива цикли, съответни на крайни области,

също е прост цикъл или обединение на ребрено непресичащи се цикли, затварящи областите $f_1, f_2, ..., f_t$.

Тъй като всеки прост цикъл затваря точно една област, то $C_1, C_2, ..., C_{k-1}$ са линейно независими, т.е. никой цикъл C_i, $1 \leq i \leq k-1$, не се получава след сумиране по $mod\ 2$ на други от тези цикли.

Освен това, всеки произволен цикъл C в графа G, затварящ крайните области $f_1, f_2, ..., f_t$, може да се представи така:

$$C = C_1 \oplus C_2 \oplus ... \oplus C_t.$$

Например простият цикъл $C = ADCC_1D_1A_1A$ в графа от последния чертеж загражда областите f_3 и f_4 и следователно $C = C_3 \oplus C_4$, където C_3 и C_4 са простите цикли, затварящи областите f_3 и f_4.

С други думи, в планарния граф G простите цикли $C_1, C_2, ..., C_{k-1}$, съответни на крайните области, образуват базис в подпространството на простите цикли в графа G.

От казаното очевидно следва верността на теоремата на Ойлер (теорема 7.4), която ще формулираме още веднъж на езика на графите.

▷ *ТЕОРЕМА 7.5. (Формула на Ойлер) Ако G е свързан планарен граф с n върха, m ребра и k области, то*

$$n - m + k = 2.$$

Доказателство: От направените по-горе разсъждения следва, че цикломатичното число μ на графа G е равно на $k-1$. По дефиниция $\mu = m - n + 1$ (G е свързан), откъдето

$$m - n + 1 = k - 1 \implies$$
$$n - m + k = 2,$$

което трябваше да се докаже. ◁

От теоремата на Ойлер се получават някои интересни следствия.

СЛЕДСТВИЕ 1. Съществуват точно 5 вида правилни многостени — тетраедър, куб, октаедър (осем триъгълнни стени), додекаедър (дванадесет петоъгълни стени) и икосиедър (двадесет триъгълни стени).

Правилен многостен наричаме такъв изпъкнал многостен, за който през всеки връх минават равен брой равни ръбове и всяка стена има равен брой равни ръбове.

Доказателство: Нека всяка стена има r на брой ръба и през всеки връх минават q на брой ръба. Тогава

$$n.q = 2m \quad \text{и} \quad k.r = 2m,$$

$$n = \frac{2m}{q} \quad \text{и} \quad k = \frac{2m}{r}.$$

От формулата на Ойлер $n - m + k = 2$, следователно

$$\frac{2m}{q} - m + \frac{2m}{r} = 2,$$

$$m(2r + 2q - qr) = 2qr,$$

откъдето се получава

$$m = \frac{2qr}{2r + 2q - qr},$$

$$n = \frac{4r}{2r + 2q - qr},$$

$$k = \frac{4q}{2r + 2q - qr}.$$

Тъй като m, n и k са естествени числа, то

$$2q + 2r - qr > 0, \quad \text{т.е.} \quad (q-2)(r-2) < 4.$$

Целите решения $r \geq 3$ и $q \geq 3$ на това неравенство са двойките $(3,3)$, $(4,3)$, $(3,4)$, $(5,3)$ и $(3,5)$, което съответства на петте вида правилни многостени — тетраедър, куб, октаедър, додекаедър и икосиедър.

Всички правилни многостени са били известни още в древността. На тях е посветена заключителната $XIII$ книга "Начала" на Евклид. Правилните многостени се наричат още Платонови тела. За древните гърци платоновите тела олицетворяват: тетраедърът — огънят, кубът — земята, икосиедърът — водата, октаедърът — въздухът, а петият многостен — додекаедърът — символизира цялата вселена.

СЛЕДСТВИЕ 2. Всеки изпъкнал многостен има поне една триъгълна, четириъгълна или петоъгълна стена.

Доказателство: Да допуснем, че всяка стена има най-малко 6 ръба, т.е. $6k \leq 2m$.

Ясно е, че през всеки връх минават поне 3 ръба, т.е. $3n \leq 2m$.

Като съберем тези две неравенства, получаваме $6k+3n \leq 4m$.

От формулата на Ойлер $3k + 3n = 3m + 6$. Като заместим в последното неравенство, получаваме

$$3k \leq m - 6.$$

От последното неравенство и неравенството $3n \leq 2m$, намираме

$$3k + 3n \leq 3m - 6.$$

Но $3k + 3n = 3m + 6 \Longrightarrow 3m + 6 \leq 3m - 6$, което е невъзможно.

Следователно един изпъкнал многостен не може да няма поне една триъгълна, четириъгълна или петоъгълна стена.

СЛЕДСТВИЕ 3. Ако G е прост планарен граф с m ребра и $n \geq 3$ върха, то $m \leq 3n - 6$.

Доказателство: [3] Нека $F = \{f_1, f_2, \ldots, f_k\}$ е множеството от области на графа G. Под степен $d(f_i)$ на областта f_i ще разбираме броя на ребрата, участващи в границата на f_i, като ребрата-мостове се броят по два пъти (реброто e се нарича мост, когато $G - e$ има повече компоненти от G). Отчитайки аналогията между определенията за степен на връх и степен на област, можем да запишем от теорема 1.1 а)

$$\sum_{f_i \in F} d(f_i) = 2m.$$

Тъй като G е прост граф и $n \geq 3$, то $d(f_i) \geq 3$ за $\forall i$. Следователно

$$\sum_{f_i \in F} d(f_i) \geq 3k \quad \text{и} \quad 2m \geq 3k \implies k \leq \frac{2}{3}m.$$

Оттук и формулата на Ойлер

$$n - m + \frac{2}{3}m \geq 2 \quad \text{или} \quad m \leq 3n - 6.$$

СЛЕДСТВИЕ 4. Графите K_5 и $K_{3,3}$ (вж. черт. 1.17) са непланарни.

Доказателство: 1. За K_5 имаме $n = 5$, $m = 10$. Да допуснем, че той е планарен. Тогава от предишното следствие 3 се получава

$$m = 10 \leq 3n - 6 = 3.5 - 6 = 9.$$

Получаваме противоречие, следователно K_5 е непланарен.

2. За графа $K_{3,3}$ имаме $n = 6$, $m = 9$. Да допуснем, че $K_{3,3}$ е планарен. Тогава от формулата на Ойлер $n - m + k = 2$, за k получаваме

$$k = 9 - 6 + 2 = 5, \tag{7.1}$$

т.е. броят на областите в $K_{3,3}$ е 5.

От друга страна, в $K_{3,3}$ няма цикли с дължина, по-малка от 4. Следователно, степента на всяка област $d(f_i) \geq 4$, откъдето

$$2m = \sum_{i=1}^{k} d(f_i) \geq 4k, \quad \text{т.е.}$$

$$k \leq \frac{2}{4}m, \quad \text{т.е.} \quad k \leq 4. \tag{7.2}$$

От (7.1) и (7.2) стигаме до противоречие. Следователно графът $K_{3,3}$ не е планарен.

СЛЕДСТВИЕ 5. Ако G е прост планарен граф, поне един негов връх е от степен ≤ 5.

Доказателство: Нека простият планарен граф G има n върха и m ребра. Да допуснем, че степента на всеки връх е > 5. От теорема 1.1 следва $2m \geq 6n$, т.е. $m \geq 3n$. Съгласно следствие 3 обаче, $m \leq 3n - 6$. От последните две неравенства стигаме до противоречие. Следователно трябва да отхвърлим допускането, твърдението в следствието е вярно.

Ще разгледаме няколко характеризации за планарност.

3. Характеризация на Куратовски-Понтрягин за планарност

За целта ще бъде необходимо да въведем още едно понятие *хомеоморфни графи.*

Нека $e_1 = (x, y)$ и $e_2 = (y, z)$ са две ребра инцидентни с върха y, чиято степен е 2. Такива ребра се наричат *последователни ребра.* Да разгледаме следните две операции.

Операция 1. Отстраняване на върха y и замяна на ребрата e_1 и e_2 с реброто (x, z).

Операция 2. Добавяне нов връх y на реброто (x, z), т.е. замяна на реброто (x, z) с две ребра (x, y), (y, z).

Два графа G_1 и G_2 се наричат *хомеоморфни*, ако са изоморфни или стават изоморфни в резултат на прилагане на горните две операции.

Очевидно, ако графът G е планарен, то всеки хомеоморфен на него граф също е планарен. Тъй като K_5 и $K_{3,3}$ както вече доказахме са непланарни, следва, че планарният граф не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Обратното твърдение също се оказва вярно. Л. Понтрягин (без да публикува) и в последствие Куратовски са доказали следното твърдение, характеризиращо планарните графи.

▷ *ТЕОРЕМА 7.6. (Теорема на Куратовски) Графът G е планарен тогава и само тогава, когато не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$.* ◁

Доказателството на тази теорема може да намерите в [4], а също така и в [30].

ПРИМЕР 7.1. Да разгледаме графите от черт. 1.19:

Черт. 1.19

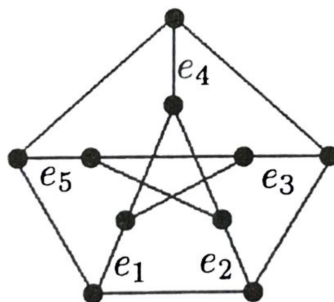
Графът от черт. 1.19 б) е подграф на графа от черт. 1.19 а). При това, очевидно в резултат на прилагане на *операция 1* за ребрата (x, y) и (y, z), т.е. отстраняване на върха y, получаваме граф, който е изоморфен на $K_{3,3}$. Следователно графът от черт. 1.19 а) е непланарен.

4. Характеризация на Вагнер, Харари и Тат за планарност

Съществува и следната характеризация на планарни графи, получена от авторите на [31] и [32]:

▷ *ТЕОРЕМА 7.7. Графът G е планарен тогава и само тогава, когато не съдържа подграф, който може да бъде свит до графа K_5 или до графа $K_{3,3}$.* ◁

Операцията свиване дефинирахме в параграф 1.2. Да разгледаме следния граф (наречен *граф на Петерсен* - черт. 1.20).



Черт. 1.20

Този граф не съдържа подграф, изоморфен на K_5 или $K_{3,3}$. Следователно, за да използваме критерия на Куратовски с цел да покажем непланарността на този граф, трябва да търсим подграф хомеоморфен на K_5 или на $K_{3,3}$ и по-точно подграф, който става изоморфен на K_5 или $K_{3,3}$, в резултат на прилагане на операциите 1 и 2, които дефинирахме по-горе.

От формулирания в теорема 7.7 критерий, непланарността на графа на Петерсен следва очевидно, тъй като след свиване на ребрата e_1, e_2, e_3, e_4, e_5, се получава графът K_5 (вж. черт. 1.17).

5. Характеризация на Маклейн

▷ *ТЕОРЕМА 7.8. Графът G е планарен тогава и само тогава, когато в него съществува такова множество от базисни цикли, че никое ребро не участва в повече от два от тези цикли.* ◁

Верността на условието за необходимост в теоремата следва от това, че циклите в планарния граф G съответни на крайните области, образуват базис и никое ребро на графа не участва в повече от два такива цикъла.

Доказателство на достатъчността е изложено в [33].

Съществуват и други характеризации на планарните графи, свързани с понятието двойнствени графи.

6. Двойнствени графи

Двойнствеността е определена за пръв път от Уитни [34]. По същество графът G_2 се явява *двойнствен (дуален)* на графа G_1, ако съществува биективно изображение между техните ребра, така че едно множество ребра в G_2 е цикличен вектор тогава и само тогава, когато съответното му множество ребра в G_1 е вектор-просто разрязващо множество в G_1.

С други думи, за да бъдат G_2 и G_1 двойнствени, е достатъчно на базисните вектори в подпространството на циклите в G_2 да съответстват вектори, образуващи базис в подпространството на разрезите в G_1 (вж. параграф 1.4).

Ще формулираме следните твърдения, свързани с двойнствеността:

▷ *ТЕОРЕМА 7.9. Ако G_2 е двойнствен на графа G_1, то и графът G_1 е двойнствен на графа G_2.* ◁

▷ *ТЕОРЕМА 7.10. Ако G_2 е двойнствен на G_1, то всеки прост цикъл в графа G_2 е съответен на просто разрязващо множество в G_1 и обратно.* ◁

▷ *ТЕОРЕМА 7.11. Ако G_2 е двойнствен на G_1, то рангът на единия граф е равен на цикломатичното число на другия.* ◁

▷ *ТЕОРЕМА 7.12. Ако графът G има двойнствен граф, то всеки негов ребрено породен подграф също има двойнствен граф.* ◁

▷ *ТЕОРЕМА 7.13. Ако графът G има двойнствен граф, тогава всеки хомеоморфен на G граф също има двойнствен граф.* ◁

Следните няколко твърдения разкриват връзката планарност — двойнственост:

▷ *ТЕОРЕМА 7.14. Ако G е планарен граф, то G има двойнствен граф.* ◁

Оказва се, че обратното твърдение на формулираното в теорема 7.14 също е вярно, т.е. можем да формулираме следната по-обща теорема:

▷ *ТЕОРЕМА 7.15. Графът G има двойнствен тогава и само тогава, когато е планарен.* ◁

Ще докажем необходимостта на теорема 7.15. Да допуснем, че графът G има двойнствен граф и G не е планарен. Тогава, съгласно теорема 7.6 (теорема на Куратовски), G ще съдържа подграф H, хомеоморфен на K_5 или $K_{3,3}$. Тъй като G има двойнствен граф, съгласно теорема 7.12, подграфът H ще има двойнствен граф, откъдето, съгласно теорема 7.13, K_5 или $K_{3,3}$ трябва да има двойнствен граф. Ще докажем, че това е невъзможно, т.е. че K_5 и $K_{3,3}$ нямат двойнствени, с което ще получим противоречие вследствие допускането, че G не е планарен.

ЛЕМА 7.1. Графът K_5 няма двойнствен граф.

Доказателство: Да допуснем противното, а именно графът K_5 има двойнствен граф G. Тъй като простите разрязващи множества в K_5 се състоят от 4 или 6 ребра (проверете), то в графа G всички прости цикли ще бъдат с дължина 4 или 6 — четна дължина. Но тогава G ще бъде биполярен (2-хроматичен) граф (вж. теорема 2.1).

Тъй като биполярен граф с 6 или по-малко върха има най-много 9 ребра, би трябвало графът G да има поне 7 върха (*).

Тъй като в K_5 всички цикли са с дължина повече от 2, то степента на всеки връх в графа G също е повече от 2 (**).

От изводите (*) и (**) и теорема 1.1 следва, че графът G трябва да има поне $\frac{7.3}{2}$ ребра, т.е. повече от 10 ребра, което противоречи на условието, че G има 10 ребра, толкова колкото са и ребрата на двойнствения му граф K_5. Следователно графът K_5 няма двойнствен. ◁

ЛЕМА 7.2. Графът $K_{3,3}$ няма двойнствен.

Доказателство: 1. Графът $K_{3,3}$ няма прости разрязващи множества, състоящи се от две ребра.

2. Графът $K_{3,3}$ има цикли с дължина само 4 или 6.

3. Графът $K_{3,3}$ има 9 ребра.

Да допуснем, че графът $K_{3,3}$ има двойнствен граф G. От 1, 2 и 3 следва:

4. Графът G няма паралелни ребра (цикли с дължина 2).

5. Графът G не съдържа прости разрязващи множества с по-малко от 4 ребра, т.е. степента на всеки връх в G е не по-малка от 4.

От 4 и 5 следва, че G има поне 5 върха, всеки от които е от степен не по-малка от 4. Тогава, съгласно теорема 1.1, G трябва да има не по-малко от $\frac{5.4}{2} = 10$ ребра (*).

От 3 и допускането, че G е двойнствен за $K_{3,3}$, следва, че G има 9 ребра (**).

От (*) и (**) стигаме до противоречие, което доказва верността на лемата. ◁

С това доказателството на необходимостта в теорема 7.15 е завършено. Доказателството на теоремата е предложено от Парсънс в [35].

7. Оцветявания

В този параграф вече видяхме, че равнинният граф G определя равнинна карта, състояща се от самия граф G и страните (областите), които графът определя. Още през XIX век е формулирана следната задача:

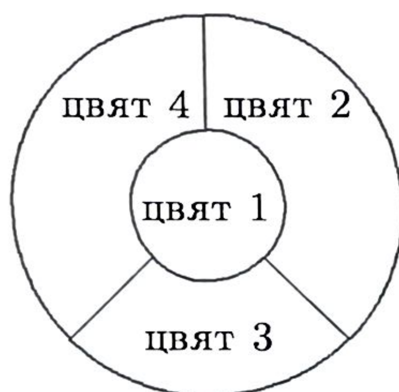
Може ли равнинна карта да бъде оцветена с 4 цвята така, че никои две страни, които имат общо ребро в границите си, да не са оцветени в един и същ цвят?

За пръв път през 1852 г. Франсис Гътри формулира тази задача, известна като *хипотеза за четирите цвята.* Повече от 100 години хипотезата за четирите цвята, въпреки многобройните опити, остава недоказана.

Разпространявана също от Де Морган (учител на Гътри), хипотезата става по-известна едва през 1878 г., когато Кейли я предлага на Лондонското математическо дружество. Веднага след това се появяват многобройни "доказателства" и техни опровержения, за да се стигне до 1976 г., когато Апел и Хакен в

[36] доказват верността на хипотезата, като за целта използват и компютри. Ние няма подробно да разглеждаме проблема за четирите цвята — обзор на проблема с исторически подробрости и неговото развитие могат да се намерят в [37] — [40].

Очевидно е, че хипотезата за четирите цвята не може да се усили, т.е. в общия случай три цвята не са достатъчни, за да бъде оцветена правилно (никои две съседни страни да не са оцветени с един и същ цвят) равнинна карта. Например равнинната карта изобразена долу, не може да се оцвети с по-малко от четири цвята.



8. Оцветяване на върховете

Да разгледаме графа G, определен по следния начин:

1. Върхове на графа G са съответните страни (области) на равнинна карта.

2. В графа G съществува ребро между два негови върха тогава и само тогава, когато върховете са съответни на съседни страни от картата (страни, чиито граници имат общо ребро).

Ясно е, че хипотезата за четирите цвята, формулирана за графа G, е: Може ли върховете на планарния граф G да се оцветят правилно с четири цвята, т.е. никои два съседни върха в графа G да не са оцветени с еднакъв цвят?

Ще разгледаме някои резултати, свързани с проблема за правилно оцветяване върховете на един граф.

Върхово k-оцветяване на G — това е оцветяване на върховете му с k различни цвята.

Правилно (върхово) k-оцветяване ще наричаме оцветяване, при което никои два върха не са оцветени с един и същ цвят.

Графът G се нарича *k-оцветим* (върхово), ако съществува правилно (върхово) k-оцветяване.

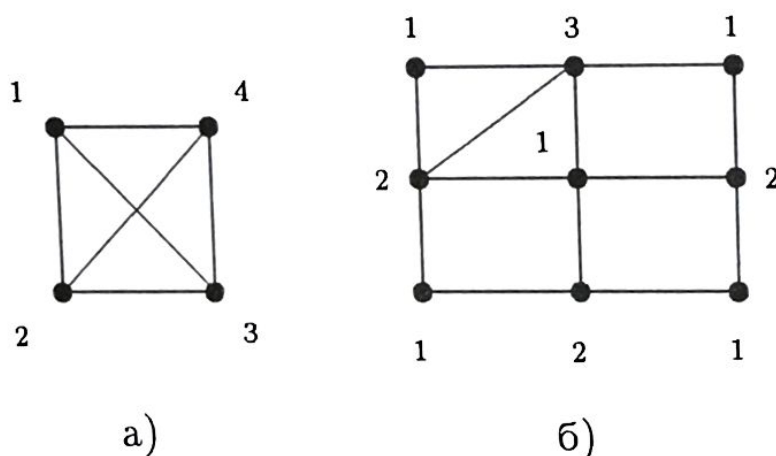
В последствие думата "върхово" ще пропускаме и ще разглеждаме прости графи.

Хроматично число $\text{æ}(G)$ на графа G ще наричаме минималното число k, за което G е k-оцветим.

Графът G ще наричаме *k-хроматичен*, ако $\text{æ}(G) = k$.

В бъдеще за краткост вместо "правилно върхово k-оцветяване", ще казваме "k-оцветяване", аналогично за графа, че е "k-оцветим".

Графът от черт. 1.21 а) има хроматично число, равно на 4, а този от черт. 1.21 б) е с хроматично число 3.



Черт. 1.21

Очевидно е, че хроматичното число на пълния граф K_n е $\text{æ}(K_n) = n$.

Освен това е ясно, че k-оцветяването на графа $G = (V, A)$ поражда разбиване на множеството от върхове V, на независими множества $V_1, V_2, \ldots, V_k$, при което на всяко подмножество V_i е присвоен цвят i. Обратното също е вярно.

Редица реални проблеми се свеждат до задачата за намиране на минимален брой цветове за правилно оцветяване върховете на един граф, т.е. до определяне на хроматичното число на графа.

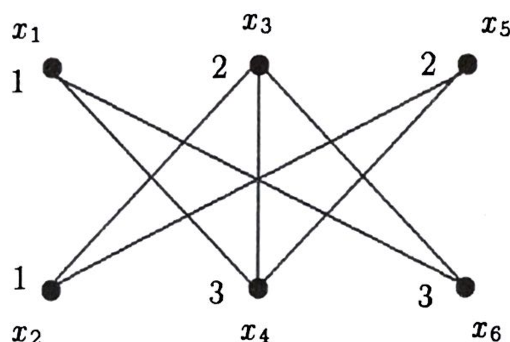
Един възможен подход да се атакува този проблем е следният "жаден алгоритъм":

Подреждат се върховете на графа в произволен ред, например $x_1, x_2, \ldots, x_n$ и в този ред се оцветяват един след друг, като стремежът е да се използват колкото се може по-малко цветове, т.е. оцветяваме върха x_1 с цвят 1; след това оцветяваме x_2 с цвят 1, ако x_2 не е съседен на x_1 и с цвят 2 — в противен случай; и т.н. Оцветяваме всеки следващ връх с възможно най-малкия (като номер) цвят.

Очевидно е, че предложената процедура не винаги води до оптимално решение, т.е. в зависимост от начина, по който са

подредени върховете, могат да се получат оцветявания с различен брой цветове. С други думи, прилагането на "жадния алгоритъм" води до оцветяване, в което може да се използват повече цветове от минимално необходимите. Ще илюстрираме казаното с пример.

ПРИМЕР 7.2. Да разгледаме следния граф G:



При избраната номерация на върховете прилагането на "жадния алгоритъм" води до:
- оцветяваме x_1 с цвят 1;
- оцветяваме x_2 с цвят 1;
- оцветяваме x_3 с цвят 2;
- оцветяваме x_4 с цвят 3;
- оцветяваме x_5 с цвят 2;
- оцветяваме x_6 с цвят 3.

С други думи, използват се 3 цвята, което не е минималния брой цветове. Тъй като графът е биполярен, множеството върхове $V_1 = \{x_1, x_3, x_5\}$ и множеството върхове $V_2 = \{x_2, x_4, x_6\}$ могат да се оцветят съответно с цвят 1 и цвят 2.

ЗАДАЧА 7.2. Подредете върховете на граф G така, че "жадният алгоритъм" да използва само $k = æ(G)$ цвята.

Очевидно, ако в "жадния алгоритъм" използваме метода *търсене с връщане назад*, ще стигнем до оптимално решение така:

На всяка стъпка се опитваме към вече оцветената част да добавим още един връх, оцветен с някои от използваните до момента цветове, ако това е невъзможно, оцветяваме текущо оцветената част по всички възможни начини с използване на същите цветове, като след всяко такова оцветяване правим опит да добавим новия връх (без да се използва нов цвят). Ако се окаже, че във всички случаи това е невъзможно, увеличаваме броя на цветовете с единица.

Предложеното решение на проблема се основава на факта, че ако част от върховете не могат да се оцветят с k цвята, то очевидно и всички върхове на графа не могат да се оцветят с k цвята.

Ясно е, че предложеният алгоритъм работи бавно. За практически нужди е целесъобразно отказването от идеята за минимален брой цветове и подмяната ѝ с оцветяване с 4 цвята, което винаги е възможно.

И в този си вариант задачата може да се реши с рекурсивно реализирано търсене с връщане назад.

ЗАДАЧА 7.3. Приложете "жадния алгоритъм", като използвате търсене с връщане назад за графа от пример 7.2.

В случаите когато размерите на графа са големи (много върхове и ребра), точните алгоритми за оцветяване работят твърде бавно. В тези случаи се използват евристични алгоритми, даващи решение близко до оптималното. Една такава евристична процедура е следната [41] и [42]:

Върховете на графа се подреждат по ненарастване на степените, т.е. колкото по-голяма е степента на един връх, толкова по-наляво в редицата е този връх. Първият връх се оцветява с цвят 1. След това в списъка от върхове (отляво надясно) със същия цвят се оцветява всеки връх, несъседен с друг, вече оцветен с този цвят връх.

След това се връщаме към първия в списъка неоцветен връх, оцветяваме го с цвят 2 и отново отляво надясно оцветяваме с цвят 2 всеки неоцветен с този цвят връх. Аналогично продължаваме процедурата, оцветявайки с цвят 3, цвят 4 и т.н., докато не бъдат оцветени всички върхове.

Очевидно, броят на използваните от алгоритъма цветове не винаги ще съвпада с хроматичното число на графа, а ще бъде едно приближение към него. Практиката показва, че този евристичен алгоритъм, работещ в реално време, ефективно дори при големи размери на графа дава добър резултат, близък до оптималния. Една прост модификация на описания по-горе алгоритъм е следната:

След като завърши оцветяването на всички възможни върхове в даден цвят, оставащите неоцветени върхове отново се подреждат по ненарастване на "остатъчните" им степени — степените в графа, който се получава от дадения, след отстраняване на оцветените върхове и инцидентните с тях ребра.

Ще формулираме две практически задачи, еквивалентни на задачата за оцветяване на върховете с минимален брой цветове [42] и [43]:

ЗАДАЧА: (съставяне на графици за прегледи, проверки и т.н.)

В календарното планиране прегледите, проверките и т.н. обикновено се представят във вид на времеви интервали. Нека на всеки преглед съпоставим връх на граф, като в този граф два върха

са свързани с ребро тогава и само тогава, когато съответните проверки не трябва да се осъществяват едновременно. Тогава задачата за съставяне на оптимален по отношение "загуба време" график е очевидно еквивалентна на задачата за оптимално оцветяване върховете на графа (минимален брой цветове).

ЗАДАЧА: (оптимално разпределение на ресурси)

Често в практиката за изпълнение на n работи (действия) трябва да се разпределят m налични ресурси — s_1, s_2, ..., s_m. Всяка от работите се изпълнява за еднакъв отрязък от време и освен това изпълнението на i-тата работа, изисква подмножеството S_i от ресурси.

Да построим граф G по следния начин: на всяка работа (действие) съответства връх от графа, като ребро между върховете на графа G съществува тогава и само тогава, когато за изпълнението на i-тата и j-тата работа се изисква използването на един и същ ресурс, т.е. $S_i \cap S_j \neq \emptyset$.

С други думи, i-тата и j-тата работа не могат да се изпълняват едновременно (ресурсите могат да бъдат машини, хора и т.н.). Очевидно е, че всяко оцветяване на графа G представлява някакво разпределение на ресурсите, свързано с изпълнението на работите, като върховете с един цвят се интерпретират като действия, които могат да се извършват едновременно. Оптималното използване на ресурсите, представляващо извършването на всички n действия за минимално време, е еквивалентно на върхово оцветяване в G с минимален брой цветове.

Ще докажем две теореми, свързани с върхово оцветяване в графи:

▷ *ТЕОРЕМА 7.16. Всеки прост граф G е $\Delta + 1$ оцветим.*

Доказателство: Да припомним, че с Δ бележим максималната степен на връх в графа G. Ще дадем процедура за оцветяване, при която не се използват повече от $\Delta + 1$ цвята:

Да вземем произволен връх v и да го оцветим в един произволен цвят измежду дадените $\Delta + 1$ цвята. Да изберем след това произволен неоцветен връх, например v_1 и го оцветим с цвят, който не е използван при оцветяването на неговите съседи. Последното е възможно, тъй като $d(v_1) \leq \Delta$ и следователно на върховете съседни с v_1, могат да са присвоени най-много Δ цвята. Тази процедура се повтаря, докато не се оцветят всички върхове.

Очевидно процедурата правилно оцветява върховете на графа и никога няма да използва $\Delta + 2$ цвята. ◁

Един възможен подход при оцветяване върховете на граф е следният: Ако е известен подграф G', който е оцветим с малко цветове, в частност да предположим с $æ(G')$ цвята, можем да оцветим оптимално върховете на G' и тогава да прилагаме предишната процедура за $\Delta + 1$-оцветяване.

В случая, когато G' е породен подграф на G и всеки подграф H, $G' \subset H \subset G$, $V(G') \neq V(H)$ съдържа връх $u \in V(H) - V(G')$, за който $d(u) \leq k$, тогава

$$æ(G) \leq \max\{k + 1,\ æ(G')\}.$$

Често пъти задачата за оцветяване върховете на граф може да се сведе до оцветяване на негови подграфи. Това се прави в случаите, когато графът е несвързан — оцветяват се оптимално отделните негови компоненти и ако се налага, чрез промяна означенията се обединяват тези оцветявания. Така може да се постъпи и когато в графа има разрязващ връх или по-общо, когато G съдържа пълен подграф, без върхове на който, графът не е свързан.

ЗАДАЧА 7.4. Ако G е пълен граф или прост цикъл с нечетна дължина, да се определи $æ(G)$.

Отг.: $æ = \Delta + 1$.

Интересно е, че за графите различни от разгледаните в задача 7.4, хроматичното число $æ \leq \Delta$. Този резултат е получен от Брукс в [44], като приведеното долу доказателство е дадено от Мелников и Визинг в [45].

▷ *ТЕОРЕМА 7.17. (Брукс) Нека G е прост свързан граф, който не е прост цикъл с нечетна държина и не е пълен граф. Тогава $æ(G) \leq \Delta$.*

Доказателство: При $\Delta \leq 2$ теоремата очевидно е вярна. Нека $\Delta \geq 3$ и допуснем противното, т.е. съществува граф G, който не е пълен и за който $\Delta \geq 3$ и $æ(G) = \Delta + 1$. Нека $G = (V, A)$ е графът с минимален брой върхове, притежаващ тези свойства.

Нека $v_0 \in V$, а G' е графът, получен от G след отстраняване на върха v_0. От начина, по който избрахме G следва, че G' е Δ-оцветим. Следователно $d(v_0) \geq \Delta$ (допускането на противното ще противоречи на $æ(G) = \Delta + 1$).

Ще формулираме следните важни свойства за G' (освен цитираното вече):

Свойство 1. Във всяко Δ-оцветяване на G' върховете, съседни с v_0 се оцветяват различно.

Нека u_1, u_2, ..., u_Δ са върховете съседни с v_0 и нека тези върхове получават при оцветяването на G', съответно цветове $1, 2, \ldots, \Delta$. С $G(i,j)$ ще бележим подграфа G', породен от върховете, оцветени с цветовете i и j.

Свойство 2. Върховете u_i и u_j се намират в една и съща свързана компонента на $G(i,j)$. Това наистина е така, защото, ако допуснем противното, след замяна на цветовете i и j в компонентата, съдържаща u_i, ще получим ново Δ-оцветяване на графа G', в което u_i и u_j ще са еднакво оцветени, което противоречи на свойство 1.

Да означим с $C(i,j)$ компонентата на $G(i,j)$, съдържаща u_i и u_j.

Свойство 3. Ще докажем, че $C(i,j)$ се явява прост път от u_i до u_j (без повтарящи се върхове).

Да допуснем, че $d(u_i)$ в $C(i,j)$ е ≥ 2. Тогава върхът u_i е съседен с не по-малко от 2 върха с цвят j. Тъй като $d(u_i) \leq \Delta - 1$ в G', върхът u_i може да се преоцвети с цвят k, различен от i и j, така че в новополученото оцветяване u_i и u_j да имат еднакъв цвят, което противоречи на свойство 1, т.е. отхвърляме допускането $d(u_i) \geq 2$, следователно $d(u_i)$ в $C(i,j)$ е $= 1$.

Аналогично $d(u_j)$ в $C(i,j)$ е $= 1$.

Степените на всички останали върхове в $C(i,j)$ са равни на 2. Да допуснем противното. Нека u е първият връх в $C(i,j)$ със степен повече от 2 на пътя от u_i до u_j. Ако u е оцветен с цвят i, то той е съседен с поне 3 върха, оцветени с цвят j. Тъй като $d(u) \leq \Delta$, можем да преоцветим u с цвят $k \neq i,\ j$, при което в новото оцветяване u_i и u_j ще се окажат в различни компоненти, което противоречи на свойство 2.

С това доказателството на свойство 3 е завършено.

Свойство 4. $C(i,j)$ и $C(i,k)$ нямат общи върхове с изключение на u_i.

Да допуснем, че съществува връх $u \neq u_i$, който се явява общ връх за $C(i,j)$ и $C(i,k)$. Тогава u е оцветен в цвят i и е съседен с поне 2 върха с цвят j и два върха с цвят k. Тъй като $d(u) \leq \Delta$, съществува цвят $l \neq i,\ j,\ k$, с който може да се преоцвети u, но това разделя върховете u_i и u_j, което противоречи на свойство 2.

Да се върнем към доказателството на теоремата (ще достигнем до противоречие със свойство 4).

От начина, по който избрахме G — непълен граф с $\Delta + 1$

върха, следва че съществуват два несъседни върха, например u_1 и u_2.

Пътят $C(1,2)$ съдържа връх $u \neq u_2$ съседен с u_1. Да допуснем, че сменяме местата на цветове 1 и 3 в пътя $C(1,3)$ (напомняме, че $\Delta \geq 3$) и получаваме ново оцветяване на G', в което върхът u_1 получава цвят 3, а върхът u_3 — цвят 1. Тогава новите компоненти $C'(1,2)$ и $C'(2,3)$ съдържат общ връх $u \neq u_2$, което противоречи на свойство 4.

С това доказателството е завършено. ◁

Накрая ще дадем доказателство на теоремата за пет цвята, дадено от автора на [46]. Доказателсво на тази теорема, използващо формулата на Ойлер, може да намерите и в [12].

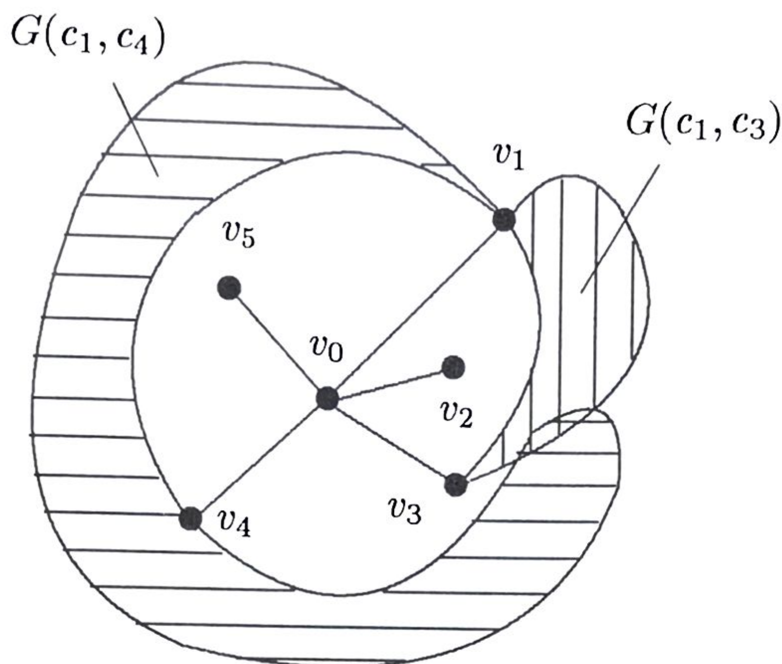
▷ *ТЕОРЕМА 7.18. Всеки планарен граф е 5-оцветим.*

Доказателство: Да допуснем, че теоремата е вярна за графи $G(V,A)$ от ред $< n$. Ще докажем, че е вярна и за графи от ред n.

От формулата на Ойлер — теорема 7.5, следствие 5, в графа G съществува връх v_0, за който $d(v_0) \leq 5$. Да разгледаме подграфа G', породен от върховете $V - \{v_0\}$.

Съгласно индуктивното допускане G' може да се оцвети с пет цвята c_1, c_2, c_3, c_4 и c_5. Ако $d(v_0) \neq 5$, то на върха v_0 в правилното 5-оцветяване на G може да се даде един от тези цветове. В противен случай на петте съседа v_1, v_2, v_3, v_4 и v_5 на v_0, могат да се присвоят различни цветове.

Да допуснем, че на v_i, $1 \leq i \leq 5$ е присвоен цвят c_i и нека върховете v_1, v_2, ..., v_5 са разположени по часовниковата стрелка спрямо v_0, както е показано на черт. 1.22.



Черт. 1.22

Нека $G(c_i, c_j)$ е подграфът на G, породен от върховете, на които е присвоен цвят c_i или c_j. Компонентата $G(c_1, c_3)$, съдържаща върха v_1, трябва да съдържа и върха v_3, тъй като в противен случай, взаимната смяна на цветовете c_1 и c_3 в тази компонента ще води до оцветяване на v_0 в цвят c_1.

Аналогично компонентата $G(c_1, c_4)$, съдържаща върха v_1 ще съдържа и върха v_4 (черт. 1.22).

Тогава в $G(c_2, c_5)$ върховете v_2 и v_5 няма да бъдат съединени с път. Това дава възможност в компонентата $G(c_2, c_5)$, съдържаща върха v_2 да сменим местата на цветовете c_2 и c_5, при което v_0 може да се оцвети в цвят c_2. По този начин се оказва, че графът G е 5-оцветим.

С това теоремата е доказана. ◁

9. Ребрено оцветяване. Хроматичен индекс

Ребрено k-оцветяване на графа се нарича присвояването на ребрата на k различни цвята.

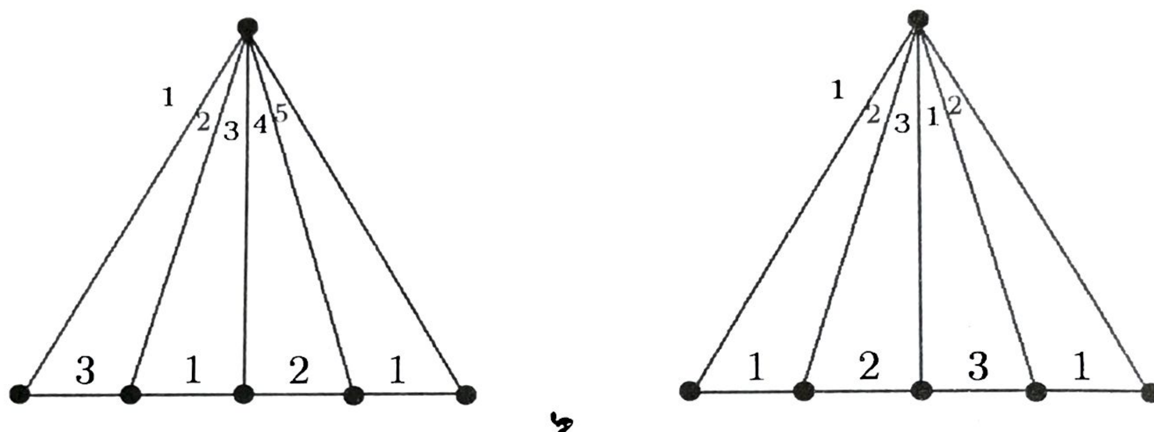
Ребреното k-оцветяване се нарича *правилно*, ако никои две съседни ребра не получават еднакъв цвят.

Графът G се нарича *ребрено k-оцветим*, ако за него съществува правилно ребрено k-оцветяване.

Хроматичен индекс или *ребрено хроматично число* $æ'(G)$ на графа G се нарича минималното число k, за което G е ребрено k-оцветим.

Графът G се нарича *ребрено k-хроматичен*, ако $æ'(G) = k$.

На черт. 1.23 е даден граф с хроматичен индекс 5. Числото приписано на всяко ребро е номер на цвета.



Черт. 1.23

Следните няколко бележки са очевидни следствия от дефинициите.

1. Хроматичният индекс (ребреното хроматично число) $æ'(G)$ очевидно е не по-малък от $\Delta(G)$, т.е.

$$æ'(G) \geq \Delta(G) = \max_{v_i \in V} d(v_i).$$

За големи класове графи горното нестрого неравенство се изпълнява като равенство. Например за биполярните графи $æ'(G) = \Delta(G)$.

2. Всяко ребрено k-оцветяване на графа $G(V, A)$ води до разбиване $(A_1, A_2, ..., A_k)$ на множеството A, където A_i е подмножеството от ребра, оцветени с цвят i. Обратното също е вярно.

3. Друга тривиална долна граница за $æ'(G)$ е следната:

Ако в G няма $\beta + 1$ независими ребра*, следва че всеки хроматичен клас има най-много β ребра. Ако броят на ребрата в G е m, тогава очевидно за $æ'(G)$ имаме

$$æ'(G) \geq \left\lceil \frac{m}{\beta} \right\rceil,$$

*Независимо множество ребра е такова, в което никои две ребра не са съседни.

където $\lceil z \rceil$ е най-малкото цяло число $\geq z$.

4. Ако K_n е пълен граф с n върха, то
а) $æ'(K_n) = n - 1$, при n — четно;
б) $æ'(K_n) = n$, при $n \geq 3$ — нечетно.

5. Тъй като всяко ребро е съседно с най-много $2(\Delta-1)$ ребра, една горна граница за $æ'(G)$ е

$$æ'(G) \leq 2\Delta - 1.$$

6. Ако $\Delta(G) > 3$, от теоремата на Брукс следва

$$æ'(G) \geq 2\Delta - 2.$$

7. Ако оцветяването $C = (A_1, A_2, ..., A_k)$ е правилно, то всяко A_i се явява сдвояване. Следователно $æ'(G)$ е минималния брой сдвоявания, на които се разбива множеството ребра на графа. Тази характеризация на хроматичния индекс се оказва полезна при доказателството на някои твърдения. Например:

▷ *ТЕОРЕМА 7.19. Ако G е биполярен граф, то* $æ'(G) = \Delta$.

Доказателство: В 1. установихме, че $æ'(G) \geq \Delta$. Тъй като множеството ребра в биполярния граф може да се разбие на Δ сдвоявания, то $æ'(G) \leq \Delta$.От последните две неравенства следва $æ'(G) = \Delta$, което трябваше да се докаже. ◁

В общия случай, Δ цвята са недостатъчни, за да се оцветят правилно ребрата на един граф.

Визинг в [47] е доказал, че за прост граф са достатъчни $\Delta+1$ цвята за правилно ребрено оцветяване.

▷ *ТЕОРЕМА 7.20. (Визинг) Ако* $G = (V, A)$ *е прост граф, то или* $æ'(G) = \Delta$ *или* $æ'(G) = \Delta + 1$ *(т.е.* $\Delta \leq æ'(G) \leq \Delta + 1$*).*

Доказателство: Ще изложим едно доказателство, дадено в [12], което се отличава със своята краткост и яснота.

Да допуснем, че вече сме използвали цветовете $1, 2, ..., \Delta + 1$ за оцветяване на всички ребра с изключение на едно. Ще приключим, ако можем да покажем, че това ребро също може да бъде оцветено.

Казваме, че един цвят *отсъства* във върха z, ако никое от ребрата, инцидентни с този връх, не е оцветено с него. Ако z

е инцидентен с $d'(z) \le d(z) \le \Delta$ оцветени ребра, то $\Delta + 1 - d'(z)$ цвята отсъстват в z. В частност във всеки връх отсъства поне един цвят. Нашата цел е да разместим цветовете и неоцветеното ребро по такъв начин, че един цвят да отсъства в двата му края, което ще ни позволи да завършим оцветяването.

Нека xy_1 е неоцветеното ребро. Нека s е цвят, който отсъства в x, и нека t_1 е цвят, който отсъства в y_1. Ще построим редица от ребра xy_1, xy_2, ... и редица от цветове t_1, t_2, ... такива, че t_i отсъства в y_i, а xy_{i+1} има цвят t_i. Да допуснем, че сме построили xy_1, ..., xy_i и t_1, ..., t_i. Има най-много едно ребро xy с цвят t_i. Ако $y \notin \{y_1, ..., y_i\}$, полагаме $y_{i+1} = y$ и избираме цвят t_{i+1}, който отсъства в y_{i+1}. В противен случай прекратяваме редицата. Тези редици трябва да завършат с най-много $\Delta(G)$ члена — нека xy_1, ..., xy_h и t_1, ..., t_h са окончателните редици. Да разгледаме двете възможни причини, които са ни принудили да прекратим редиците.

а) *Няма ребро xy с цвят t_h.* Тогава ще преоцветим всяко от ребрата xy_i, $i < h$, съответно с цвета t_i. В полученото оцветяване всички ребра са оцветени с изключение на xy_h. Обаче, тъй като t_h отсъства както в x, така и в y_h, можем да приключим, като оцветим xy_h с t_h.

б) *За някое $j < h$ реброто xy_j има цвят t_h.* Най-напред ще преоцветим всяко от ребрата xy_i, $i < j$, съответно с цвета t_i. Тогава неоцветеното ребро е xy_j. Нека $H(s, t_h)$ е подграф на G, образуван от ребрата с цветове s и t_h. Всеки връх от $H(s, t_h)$ е инцидентен с най-много две ребра в $H(s, t_h)$ (едното с цвят s, а другото — с t_h). Затова компонентите на $H(s, t_h)$ са пътища и цикли. Всеки от върховете x, y_j и y_h има най-много степен 1 в $H(s, t_h)$, затова те не могат да принадлежат на една и съща негова компонента. Тогава поне един от следните случаи е налице.

б1) *Върховете x и y_j принадлежат на различни компоненти на $H(s, t_h)$.* Да разменим цветовете s и t_h в компонентата, съдържаща y_j. Тогава s отсъства както в x, така и в y_j и затова можем да приключим, като оцветим xy_j с s.

б2) *Върховете x и y_h принадлежат на различни компоненти на $H(s, t_h)$.* Да продължим преоцветяването на ребрата, инцидентни с x, като xy_i оцветим в цвета t_i за всяко $i < h$. Така xy_h остава неоцветено. Тази промяна не включва ребра с цветове s и t_h, затова $H(s, t_h)$ не се изменя. Сега да разменим цветовете в компонентата, която съдържа y_h. Това гарантира, че s отсъства както в x, така и в y_h, затова можем да го използваме за оцветяване на реброто xy_h. ◁

Забележете, че горното доказателство дава алгоритъм за оцветяване на ребрата с не повече от $\Delta + 1$ цвята.

По-подробно изложение на предшестващите резултати читателят може да намери в Хаджииванов Н., Числа на Ремзи [14].

1.8. Алгоритми за анализ на графи

Ясно е, че изследването на редица задачи от практиката налага използването на графи. Успешното решаване на задачите изисква да се определят теоретико-графовите свойства на разглеждания проблем. Това е особено важно, защото ще даде идеи, методи и конкретни алгоритми за решаване на даден клас проблеми. Например, изследването и установяването на редица свойства на биполярните графи (параграфи 1.2 и 1.5), теоремите на Хол, Берж и Менгер, алгебричната теория и матричното представяне на графи дават теоретична основа, на която се базират и конструират алгоритми, решаващи реални проблеми от практиката.

По принцип графите, с които се описват реалните задачи, са с голяма размерност. *Изчислителната сложност* на алгоритъма, явяваща се мярка за времето на изпълнение на алгоритъма, е функция от размера на задачата, представена с входни данни. При задачи с графи под размерност на задачата може да се разбира $|V|$, т.е. броя на върховете. Сложността на алгоритъма се определя най-общо като функция f, така че $f(n)$ е равно на най-големия брой стъпки на алгоритъма за произволен граф с n върха. Под размерност на задачата може да се разбира и двойката $\langle |V|, |E| \rangle$, т.е. сложността е функция на две променливи $f(n, m)$ и е равна на най-големия брой стъпки, изпълнявани от алгоритъма за произволен граф с n върха и m ребра. При изследване сложността на алгоритъма се използва следния *символ на Ландау*

$$f(n) = O(g(n)) \Longleftrightarrow \quad \exists\ c,\ n_0 > 0 \ \text{ такива, че за } \ \forall n \geq n_0 \quad f(n) \leq c.g(n).$$

Символът $O(g(n))$ се чете "порядък не по-голям от $g(n)$". По-подробно въпросите, свързани със сложност на алгоритъм са дадени в параграф 2.4. Цитираната сложност се нарича още времева сложност за разлика от сложността по памет.

Тъй като реалните задачи са с голяма размерност, така наречените алгоритми с пълно изчерпване или не дават в реално

време решение на проблема, или го правят бавно и неефективно. При динамични системи, където бързодействието е от особена важност, се налага не просто да се намери алгоритъм, а да се намери бърз и ефективен алгоритъм.

В този параграф ще бъдат разгледани въпроси, свързани с машинното предстаяне на графи и някои основни методи за търсене в графи — търсене в дълбочина, търсене в ширина и търсене с връщане назад. Тези методи широко се използват при конструирането на конкретни оптимизационни алгоритми.

1. Машинно представяне на графи

Представянето на графите като изображение в равнината чрез точки и съединяващи ги линии е безспорно полезно, ясно и удобно за човека. Това представяне на графите обаче е безполезно и неприложимо като форма за изчислителните машини. Ще акцентираме върху няколко основни начина за представяне на информацията, свързана с един граф във вид, удобен за компютъра.

В параграф 4 на тази глава "Матрично представяне на графи" бяха дадени вече някои идеи в тази посока.

МАТРИЦА НА ИНЦИДЕНТНОСТ (ПАРАГРАФ 1.4). Това е един начин информацията за графа да се съхранява и обработва от компютър.

а) позитиви. Очевидно матрицата на инцидентност дава възможност да се извлича полезна информация за графа. Най-малко, съгласно теорема 4.3, с нейна помощ може да се определи броят на покриващите дървета в един граф и самите дървета.

б) негативи. От алгоритмична гледна точка матрицата на инцидентност не е най-добрия начин за представяне на графа, защото:

1. Необходими са $n.m$ клетки от паметта, при това голяма част от тях са заети с нули.

2. Достъпът до информацията не е удобен. За да се установи съществуването на дъга (v_i, v_j) или да се определят съседите на връх v_i, се налага (в най-лошия, неблагоприятен случай) да се прегледат всички стълбове на матрицата, което означава m стъпки.

МАТРИЦА НА СЪСЕДСТВО (ПАРАГРАФ 1.4) В много случаи това е по-добър начин за представяне на графа G.

а) позитиви. В случая въпроси от типа за съществуване на ребро (u, v) се решават с една стъпка. Освен това, както видяхме в параграф 1.4, с матрицата на съседство лесно се определят степените на върховете и се намира броят на пътищата между два върха с фиксирана дължина s. Матрицата на съседство дава възможност (параграф 1.6) за намиране на хамилтоновите вериги в графи и е полезна при търсене решения на много графови задачи.

б) негативи. Един основен негатив е, че независимо от броя на ребрата в графа, обемът на използваната памет е n^2. При малки стойности на n това неудобство може частично да се отстрани, като редът (или стълбът) на матрицата се съхранява в една машинна дума.

Втори недостатък при използване матрицата на съседство е следният: Нека P е свойство и $P(G) = 0$ да означава, че графът G не притежава това свойство, а $P(G) = 1$ — графът G притежава това свойство. Нека свойството P удовлетворява:

1. $P(G) = P(G')$, ако G и G' са изоморфни.
2. $P(G) = 0$, за произволен "празен" граф $(V, \emptyset)$ и $P(G) = 1$ за произволен пълен граф със същото множество върхове ($|V|$ е достатъчно голяма).
3. Добавянето на ребра към G не нарушава свойството P.

Оказва се, че ако P е свойство на графа, за което са изпълнени горните три условия, е в сила следната теорема:

▷ ТЕОРЕМА 8.1. [48, 49] *Всеки алгоритъм, проверяващ свойството P, т.е. изчисляващ $P(G)$ за даден граф G с помощта на матрицата на съседство, изпълнява в най-лошия случай $\Omega(n^2)$ стъпки.* ◁

В горната теорема използваният символ $\Omega(g(n))$ се дефинира по аналогия с $O(g(n))$ така:

$$f(n) = \Omega(g(n)) \iff \begin{array}{l} \exists\ c,\ n_0 > 0 \ \text{ така, че за } \ \forall n \geq n_0 \\ f(n) \geq c.g(n). \end{array}$$

Няма да се спираме върху позитивите и негативите на останалите видове матрици, които бяха разгледани в параграф 1.4.

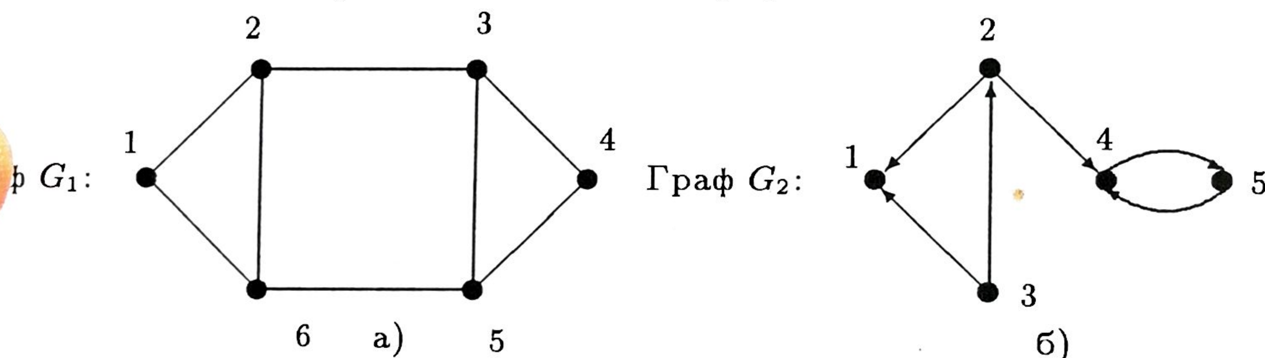
Като самостоятелно упражнение читателят може да се опита сам да направи това.

Когато графът не е достатъчно плътен, т.е. броят на ребрата m е доста по-малък от n^2, може да се използва следното представяне на графа, което е икономично по отношение на памет.

СПИСЪК ДВОЙКИ СЪОТВЕТНИ НА РЕБРАТА. В този случай обемът изискуема памет е $2m$, където m е броят на ребрата. В случая обаче, основния недостатък се състои в големия брой стъпки — порядък m, които са необходими, ако се налага да се определи множеството върхове, към които има ребра от даден връх.

Обикновено в този случай множеството двойки се подрежда лексикографически.

ПРИМЕР 8.1. Да разгледаме следните графи:



Черт. 1.24

Списъците ребра съответни на графите G_1 и G_2 са:

а)

1	2
1	6
2	3
2	6
3	4
3	5
4	5
5	6

б)

2	1
2	4
3	1
3	2
4	5
5	4

СПИСЪК НА ИНЦИДЕНТНОСТ. При този вид структуриране на данните за всеки връх $v \in V$ има списък върхове u такива, че $v \longrightarrow u$, когато графът е ориентиран или $v - u$, когато графът е неориентиран.

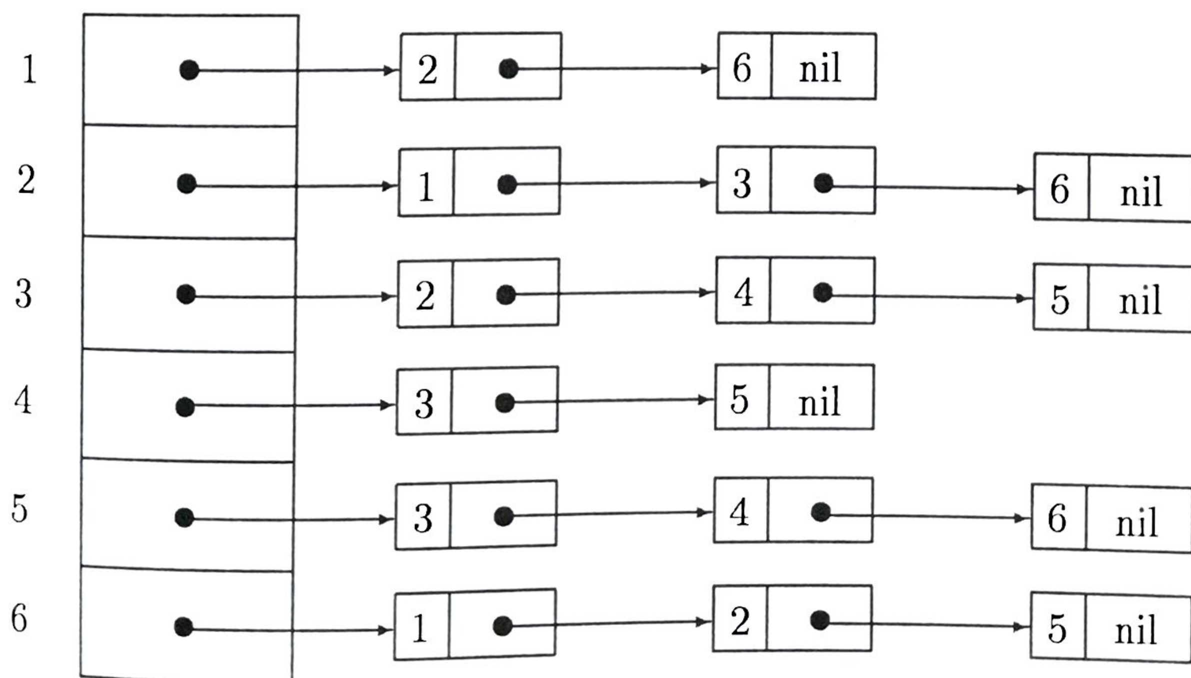
С други думи, всеки елемент от списъка на инцидентност се явява запис r, съдържащ *r.ред* и указател *r.след* за следващия запис в списъка. За последния запис в списъка *r.след*=**nil**. *НАЧАЛО* $[v]$ се явява указател за начало на списъка, съдържащ

върховете от множеството $\{u : v \longrightarrow u\}$. Неформално такъв списък ще обозначаваме със *ЗАПИС* $[v]$, а цикълът, изпълняващ дадена операция за всеки елемент u от този списък ще записваме "**for** $u \in$ *ЗАПИС* $[v]$ **do**..."

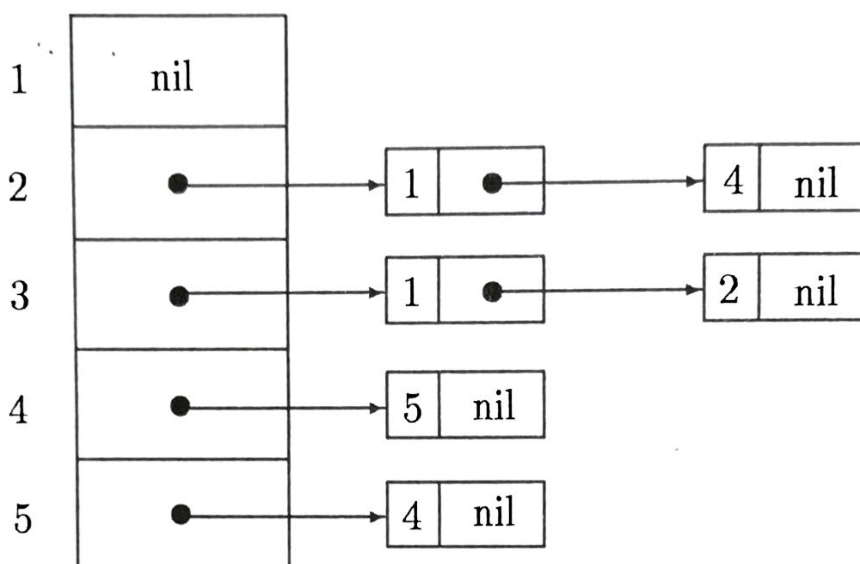
И в бъдеще ще използваме една неформална версия на езика Паскал при описването на алгоритми, взаимствана от [17].

При неориентирани графи очевидно и в този случай всяко ребро (u, v) се представя два пъти — чрез върха v в списъка *ЗАПИС* $[u]$ и чрез върха u в списъка *ЗАПИС* $[v]$.

За графите G_1 и G_2 от пример 8.1 съответните списъци на инцидентност *ЗАПИС* $[v]$ са:



a)



б)

Необходимите клетки памет при използването на списъка на инцидентност е от порядък $m + n$.

2. Търсене в дълбочина

Много практически проблеми и по-точно тяхното решаване е свързано със систематично обхождане върховете на графа така, че всеки връх да се обходи точно веднъж или k пъти, където k е ограничено от някаква "разумна" константа. Същото важи и за обхождането на ребрата в графа.

Ще дадем един метод за търсене в неориентиран граф, който е основа за проектирането на редица алгоритми в графи. Методът се нарича *търсене в дълбочина* (англ. depth first search) [50]. Общата идея на този метод е следната:

Избираме произволен връх v, от който започва търсенето. Върхът v се нарича още *корен* (алгоритъмът строи дърво, когато G е свързан) или *начален връх*. Върхът v се счита за *маркиран (разгледан) връх*.

След това избираме ребро (v, x) и преминаваме по него, за да попаднем във върха x. При това ориентираме реброто от v към x. Реброто (v, x) след тези действия се счита за *маркирано (разгледано) ребро* и се нарича *ребро на дървото*. Върхът v се нарича *баща* за върха x и се бележи с FATHER x.

В общия случай, когато се намираме в произволен връх x, има следните две възможности:

A) Всички ребра инцидентни с x вече са разгледани. Тогава се връщаме към FATHER x и продължаваме търсенето от него. Върхът x от този момент нататък се нарича *напълно сканиран (използван)*.

B) Съществуват немаркирани (неразгледани) ребра инцидентни с x. Тогава се избира едно произволно такова ребро (x, y), което се ориентира от x към y. От този момент нататък реброто (x, y) се счита за маркирано (разгледано), като са възможни следните два случая:

1. Ако не сме преминавали по-рано през върха y, т.е. y не е маркиран (разгледан), преминаваме по реброто (x, y) до върха y и продължаваме търсенето от върха y. В този случай реброто (x, y), по което сме преминали, става ориентирано, маркирано (разгледано) и се нарича *ребро на дървото*, а x =FATHER(y).

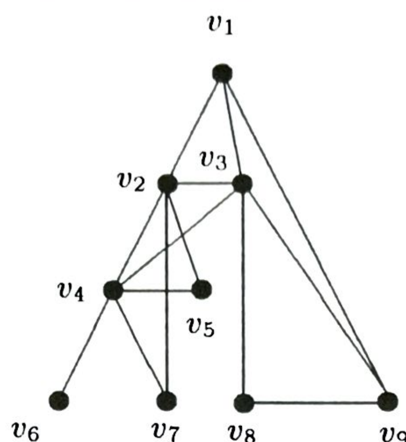
2. Ако през върха y вече сме преминавали, ориентираното ребро (x, y) се нарича *обратно ребро* и търсенето продължава по друго неразглеждано ребро инцидентно с x.

Търсенето в дълбочина завършва, когато се върнем в корена v и всички върхове са напълно сканирани.

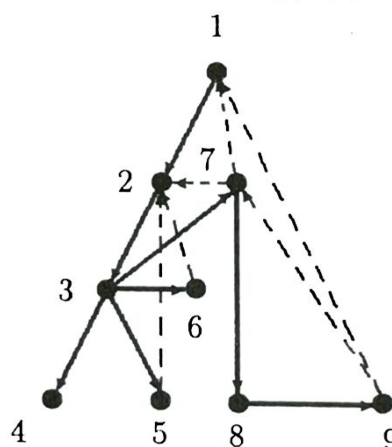
Описаното търсене в дълбочина очевидно разбива множеството ребра на графа на две подмножества — ребра на дървото и обратни ребра. Освен това е очевидно, че ребрата на дървото образуват покриващо дърво T за свързания граф G. Накрая разглеждания неориентиран граф очевидно не е задължително да бъде свързан.

Преди да илюстрираме с пример търсенето в дълбочина, че отбележим още следното. По време на търсенето в дълбочина, когато за пръв път се преминава през даден връх x, на него може да се съпостави цяло число MARK(x) така, че MARK(x) е равно на i, ако x се явява i-тия поред преминат връх. MARK(x) се нарича *дълбочина на* x и показва реда, по който се преминават върховете при търсенето в дълбочина. Въведеното понятие дълбочина на x не трябва да се бърка с понятието дълбочина на дърво, т.е. с максималното разстояние (брой ребра на дървото) между корен и лист на дървото.

ПРИМЕР 8.2. Ще илюстрираме търсенето в дълбочина в следния граф G:



Прилагането на търсене в дълбочина води до:



Черт. 1.25

КОМЕНТАР:

Числата в черт. 1.25, приписани на всеки връх v_i, са всъщност стойностите на MARK(v_i) и показват реда на преминаване през върховете при търсенето в дълбочина.

Плътните ребра са ребрата на дървото, а пунктираните ребра са обратните ребра (ребрата са ориентирани).

Обикновено получаваният ориентиран граф се бележи с $\widehat{G}$. Ребрата на дървото образуват ориентирано покриващо дърво T.

Обърнете внимание, че обхождането на графа не е единствено, тъй като ребрата инцидентни с даден връх могат да се избират в произволен ред.

Пълното сканиране на върховете, при реализираното търсене в дълбочина от черт. 1.25, се осъществява в следния ред:

$$v_6,\ v_7,\ v_5,\ v_9,\ v_8,\ v_3,\ v_4,\ v_2,\ v_1.$$

Следната рекурсивна процедура [17] дава по-формално описание на алгоритъма за търсене в дълбочина:

1 **procedure** $WG(v)$
(*търсене в дълбочина от връх v;
променливите *НЕМАРКИРАН, ЗАПИС* са глобални *)

2 **begin** разглеждане на v; *НЕМАРКИРАН* $[v]$:=**лъжа**;

3 **for** u ∈*ЗАПИС* $[v]$ **do**

4 **if** *НЕМАРКИРАН* $[u]$ **then** $WG(u)$

5 **end** (* връх v напълно сканиран *)

Когато графът не е задължително да бъде свързан, търсенето в дълбочина може да се извърши със следния алгоритъм [17]:

1 **begin**

2 **for** $v \in V$ **do** *НЕМАРКИРАН* $[v]$:=**истина**; (*инициализация*)

3 **for** $v \in V$ **do**
4 **if** *НЕМАРКИРАН* $[v]$ **then** $WG(v)$

5 **end**

От структурата на процедурата WG се вижда още, че всеки връх се разглежда не повече от един път, тъй като се разглеждат само върхове v, за които *НЕМАРКИРАН* $[v]$:=**истина** и след преминаването през този връх се изпълнява командата *НЕМАРКИРАН* $[v]$:=**лъжа** (ред 2). Тъй като алгоритъма стартира търсенето последователно от всеки все още неразгледан връх, то ще бъдат разгледани всички върхове на графа.

Сложността на този алгоритъм е $O(n+m)$.

ЗАДАЧА 8.1. Да се модифицира алгоритъма за търсене в дълбочина в произволен граф, така че да се изчисляват свързаните компоненти на графа.

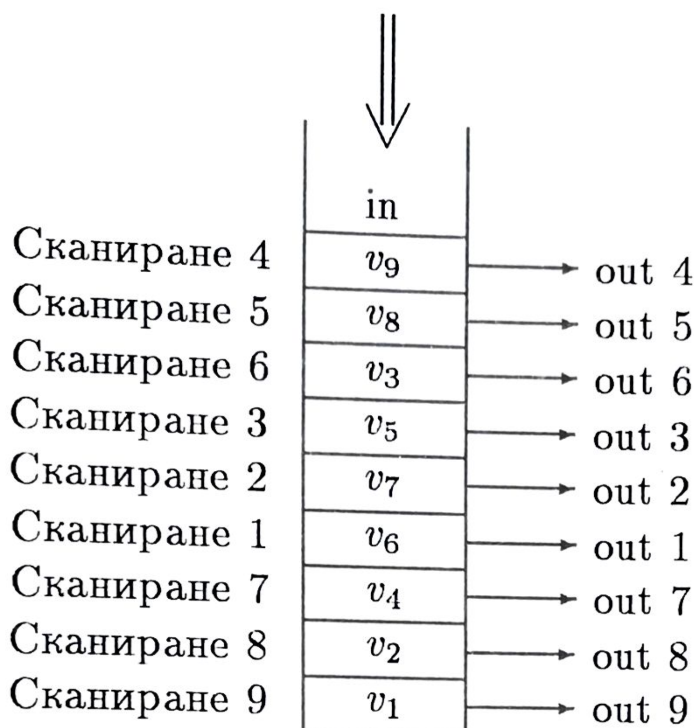
В [17] е дадена и следната нерекурсивна версия на процедурата WG, като рекурсията се отстранява по стандартния начин с използване на стек. Всеки разгледан връх се поставя в стека и се отстранява след като е напълно сканиран.

1 **procedure** $WG1[v]$
(*търсене в дълбочина с корен v; предполагаме, че в началото на търсенето $P[u]$ = указател на първия запис в списъка *ЗАПИС* $[u]$ за всеки връх u; масиви P, *НЕМАРКИРАН* — глобални *)

```
begin СТЕК:= ∅; СТЕК ⟸ v;
  разглеждане на v;
  НЕМАРКИРАН [v] := лъжа;
while СТЕК ≠ ∅ do
begin t := top (СТЕК);
  (* t-връхния елемент на стека*)
  (* намери първия нов връх в списъка ЗАПИС [t]*)
      if P[t] =nil then b :=лъжа
      else b :=not НЕМАРКИРАН [P[t] ↑ .ред];
      while b do
         begin P[t] := P[t] ↑.след;
            if P[t] =nil then b :=лъжа
            else b :=not НЕМАРКИРАН [P[t] ↑ .ред]
         end;
      if P[t] ≠nil then
      (* намерен е неразгледан връх*)
         begin t := P[t] ↑.ред; СТЕК ⟸ t;
         разглежда се t; НЕМАРКИРАН [t] := лъжа
         end
      else (* връх t е напълно сканиран*)
         t ⟸ СТЕК
         (*отстранява се връхния елемент на стека*)
   end
end
```

Търсенето в дълбочина при ориентирани графи е аналогично както за неориентирани графи с тази разлика, че преминаването по "ребрата" на графа става в съответствие с тяхната ориентация. Като следствие от това ограничение, при ориентирани графи ребрата се разбиват не на две, а на четири категории. Формално описание на съответния алгоритъм можете да намерите в [3] и др.

Реализираното търсене в дълбочина от черт. 1.25 съгласно предложената процедура $WG1[v]$ може да се илюстрира по следния начин:



Черт. 1.26

Ако в дървото T, получено при търсенето в дълбочина, съществува ориентиран път от връх v до връх u, то v се нарича *предшественик* на u, а върхът u — *потомък (наследник)* на v. Ако $v = FATHER(u)$, върхът u се нарича още *син* на v. Ясно е, че един връх може да има повече от един син.

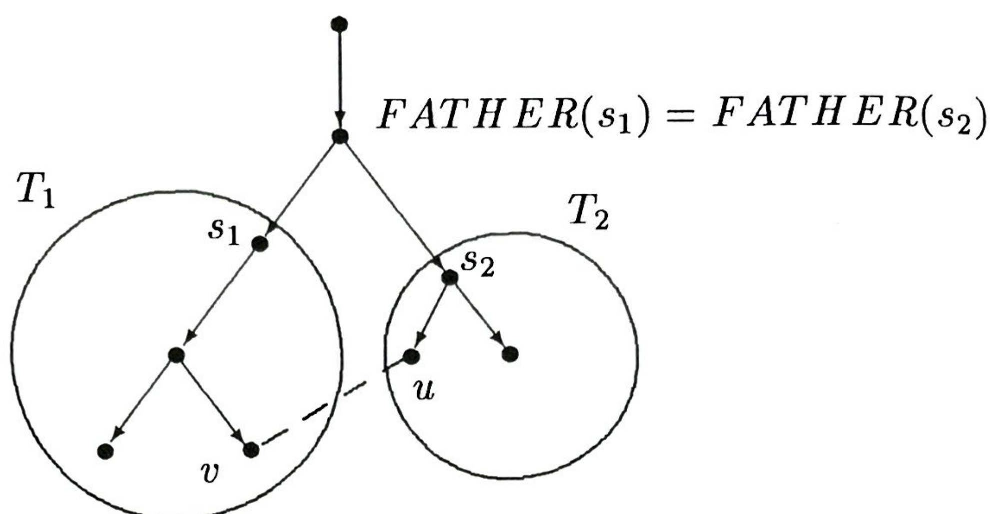
Два върха v и u се наричат *съотносими*, ако единият от тях е потомък на другия.

▷ *ТЕОРЕМА 8.2. Нека (v, u) е ребро в свързания неориентиран граф G. Тогава във всяко дърво T, съответстващо на търсене в дълбочина в този граф, върховете v и u са съотносими.*

Доказателство: Да допуснем, че съществува дърво T, в което съседните върхове v и u са несъотносими (нито v е потомък на u, нито u е потомък на v). Тогава е ясно, че съществуват два върха s_1 и s_2 такива, че:

1. $FATHER(s_1) = FATHER(s_2)$;

2. v е потомък на s_1 и u е потомък на s_2, както е показано на черт. 1.27.



Черт. 1.27

Ако T_1 и T_2 са поддърветата на T съответно с корени s_1 и s_2, без ограничение на общността да допуснем, че

$$MARK(s_1) < MARK(s_2).$$

Тогава от алгоритъма за търсене в дълбочина следва, че върховете на поддървото T_2 се разглеждат само след пълното сканиране на върха s_1, т.е. при разглеждането на върха v върхът u със сигурност е немаркиран, откъдето следва, че реброто (v, u) трябва да бъде ориентирано от v към u и да бъде маркирано съгласно алгоритъма като ребро на дървото. Противоречие, отхвърляме допускането, т.е. дървото, изобразено на черт. 1.27 не може да се получи вследствие прилагане на търсене в дълбочина (при съществуване на ребро (v, u)). ◁

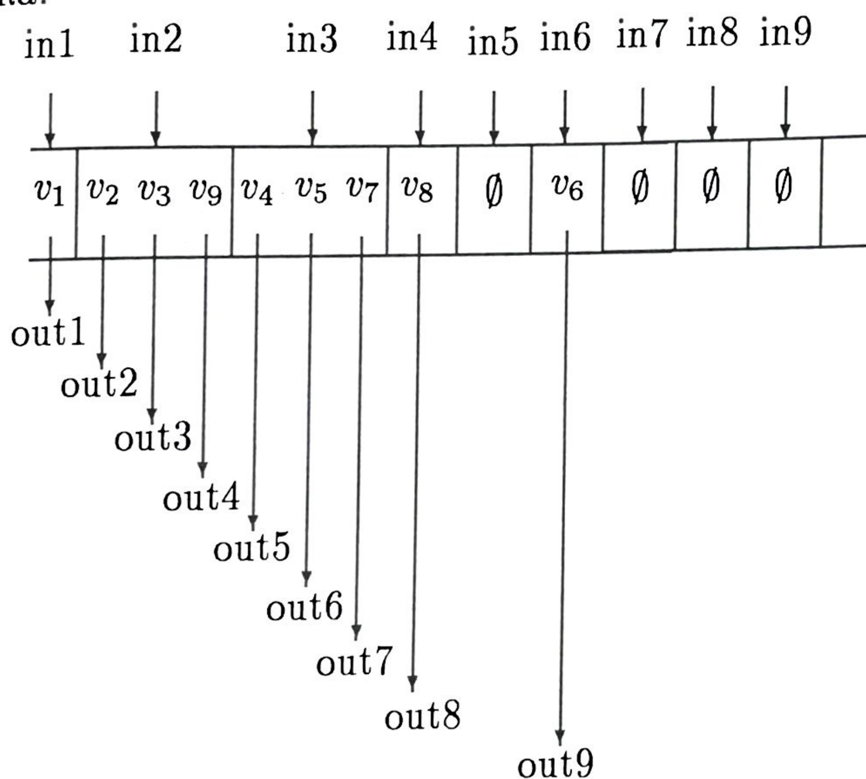
3. Търсене в ширина

Освен търсенето в дълбочина, също се използва още един метод за системно търсене в графи, наречен *търсене в ширина* (англ. breadth first search). При търсенето в дълбочина, колкото по-късно бъде посетен един връх, толкова по-рано той се

сканира. По-точно, от два върха "втория" се разглежда преди сканирането на "първия". Припомняме, че разгледаните, но все още несканирани върхове се натрупватв стек. Грубо казано, при търсенето в ширина стекът се заменя с опашка. При това модифициране колкото по-рано бъде маркиран, разгледан един връх (бъде добавен в опашката), толкова по-рано той се сканира (отстранява се от опашката).

И в този случай сканирането на върховете се извършва с помощта на обхождане на всички неразгледани съседи на този връх.

За графа от пример 8.2 търсенето в ширина може да се илюстрира така:



Процедурата търсене в ширина по-формално може да се опише така [17]:

1 **procedure** $WS[v]$
(*търсене в ширина в граф с начален връх v; променливите *НЕМАРКИРАН, ЗАПИС* са глобални *)

2 **begin**
3 *ОПАШКА*:= $\emptyset$; *ОПАШКА* $\Longleftarrow v$;
НЕМАРКИРАН $[v]$:= **лъжа**;

4 **while** *ОПАШКА* $\neq \emptyset$ **do**

5 **begin** $p \Longleftarrow$ *ОПАШКА*;

```
6        for u ∈ ЗАПИС [p] do
7            if НЕМАРКИРАН [u] then;
8                begin ОПАШКА ⇐ u;
                  НЕМАРКИРАН [u] := лъжа;
9                end
10   end
11 end;
```

Изчислителната сложност на търсенето в ширина, както и при търсенето в дълбочина, е от порядък $m + n$ (всеки връх се добавя в опашката и се отстранява от опашката точно един път, а броят итерации на цикъла от ред 6 има порядък, равен на броя на ребрата в графа).

Двата вида търсене — в дълбочина и в ширина, могат да се използват за намиране на пътища между произволни фиксирани върхове v и u в графа. За целта е достатъчно да се стартира търсенето във връх v и то да се осъществява до момента на достигане на върха u.

Търсенето в дълбочина е по-удобно от гледна точка на това, че при посещението на върха u, стекът съдържа върховете, определящи пътя от v до u (всеки връх, добавен в стека, е съседен на върха на стека).

За съжаление, полученият с търсене в дълбочина $(v - u)$ път в общия случай не е възможно най-краткия. В параграф 2.3 на следващата глава ще дадем алгоритми за търсене на най-кратки пътища.

Споменатият по-горе недостатък липсва при търсене на $(v - u)$ пътища с метода търсене в ширина. Процедурата WS може лесно да се модифицира с малка промяна (редове 7-9) [17]:

```
if НЕМАРКИРАН [u] then
    begin ОПАШКА ⇐ u; НЕМАРКИРАН [u] := лъжа;
    ПРЕДХОДЕН [u] := p
end
```

След приключване работата на така модифицираната процедура, таблицата *ПРЕДХОДЕН* съдържа за всеки маркиран връх u върха *ПРЕДХОДЕН* $[u]$, т.е. върха, от който сме попаднали в u. Най-краткият път от u до v се обозначава с последователността $u = u_1,\ u_2,\ \ldots,\ u_k = v$, където $u_{i+1} =$ *ПРЕДХОДЕН* $[u_i]$ за $1 \leq i < k$ и k се явява първия индекс i, за който $u_i = v$.

Процедурите за търсене в дълбочина и в ширина може да се използват и за намиране на покриващо дърво в свързан граф G (вж. праграф 2.3, глава 2).

4. Принципи за търсене в дървото на решенията

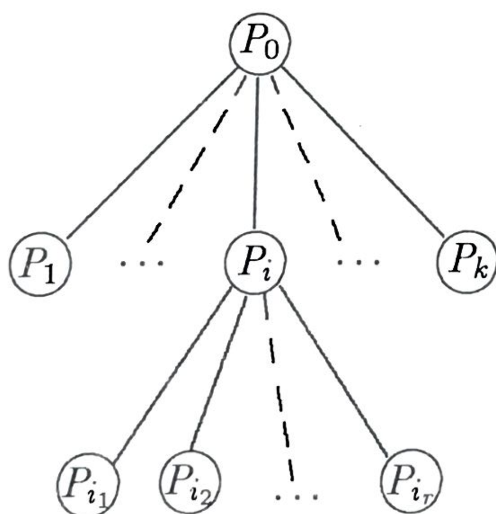
Решаването на една графова задача P_0 (начална задача) често се разбива на определен брой подзадачи P_1, P_2, ..., P_k, които в съвкупност представляват задачата P_0. Обикновено се правят опити (когато задачата P_0 не може да се реши) да се *разреши* всяка от нейните подзадачи. Под *разреши* се разбира следното:

(*) или да се намери оптимално решение;

(**) или да се покаже, че подзадачата се явява недопустима;

(***) или да се покаже, че значението на оптималното решение е по-лошо от полученото до момента най-добро решение.

Разбиването на подзадачи се описва с дърво, в което върховете са съответни на подзадачите (черт. 1.28).



Черт. 1.28

Причината за разглеждането на подзадачи на задачата P_0 обикновено е свързана с по-малката размерност на подзадачите или с притежаването от тях на структурни свойства, неприсъщи на началната задача P_0. Може да се окаже, че някоя от задачите P_i не може да се разреши, поради което и тази подзадача се разбива на нови подзадачи, както е показано на черт. 1.28.

На всеки етап множеството задачи, изискващи разрешимост, се представят с множеството от висящите върхове на дървото (върховете от степен 1). Коренът на дървото изобразява началната задача P_0.

Очевидно, ако с $\{P\}$ означим множеството от дупустими решения на една задача P, то

$$(8.1) \qquad \{P_i\} = \{P_{i_1}\} \cup \{P_{i_2}\} \cup \ldots \cup \{P_{i_r}\}.$$

Тогава очевидно

$$(8.2) \qquad \{P_0\} = \cup\{P(j) | P(j) \ - \ \text{висящ връх в дървото}\}.$$

Когато трябва да се получат (обходят) всички решения на задачата P_0 (а не само оптимално решение), трябва да се обходят решенията на всяка от подзадачите от горецитираното разбиване. В този случай е полезно да се избягва дублиране в построяваните решения, т.е. необходимо е да се разбива задачата P_i на подзадачи P_{i_1}, ..., P_{i_r} така, че

$$(8.3) \qquad \{P_{i_s}\} \cap \{P_{i_q}\} = \emptyset,$$

за всеки две подзадачи P_{i_s} и P_{i_q}, $s \neq q$.

Условието (8.3) не е необходимо за осъществяване търсене в дървото на решенията, но то е полезно, защото определя разбиване, изгодно от изчислителна гледна точка, тъй като:

1. Ако P_0 е оптимизационна задача, то оптималното решение се явява решение на една и точно една подзадача, представена с висящ връх.

2. Ако P_0 е задача, свързана с пълно обхождане, то множеството от всички нейни решения е обединение на множествата от решения на подзадачите, представени с висящи върхове, при това без дублиране.

Процесът на търсене в дървото на решенията може да се осъществи както с търсене в дълбочина, така и с търсене в ширина.

5. Използване на граници

Ясно е, че ако задачата P_0 е оптимизационна задача (търси се едно оптимално решение), без значение от типа търсене, който се използва, търсенето завършва тогава и само тогава, когато са разрешени всички подзадачи, представени с висящи върхове. За да се ускори процесът на разрешаване, за всеки от висящите върхове (подзадача) се изчисляват *долни или горни граници* (в случаите на минимизация — долна граница, а

в случая на максимизация — горна граница). Това са граници, даващи съответно най-малката или най-голямата възможна стойност на оптималното решение на подзадачата. По този начин (при задачи за минимизация), ако се окаже, че долната граница за върха (подзадачата) P_i е по-голяма от величината на най-добрия отговор, получен по-рано при търсенето, то от P_i не е необходимо по-нататъшно *разклоняване* (руск. ветвление; англ. branching). Това е така, защото в $\{P_i\}$ няма да има решение, което да бъде по-добро от текущия най-добър резултат. С други думи, от неравенствата

$$\text{"текущо оптимално решение"} < \text{Lower bound } (P_i) \leq \text{Опт.}(P_i),$$

от (8.1) и (***) следва, че подзадачата P_i се оказва автоматически разрешена.

6. Търсене с връщане назад

Да се върнем още веднъж на проблема за съществуване на хамилтонов път (виж параграф 1.6). Този проблем принадлежи към класа на т.нар. *$\mathcal{NP}$- пълни задачи* (вж. параграф 2.4). За този широк клас от фундаментални (за теория на графите, логиката, теория на числата, дискретната математика и др.) задачи, не е известен полиномиален алгоритъм — алгоритъм с брой стъпки, ограничени с полином от размерността на задачата (наличието на полиномиален алгоритъм за поне една от този клас задачи автоматично води до съществуване на полиномиални алгоритми за целия клас).

Един силов метод при търсене на хамилтонови пътища е "пълното обхождане" на всички възможности. Генерират се всички $n!$ различни последователности от върхове и за всяка се проверява дали е хамилтонов път. Очевидно това изисква не по-малко от $n.n!$ стъпки, т.е. с нарастването на n много бързо нараства броят на стъпките (по-бързо от произволна експоненциална функция от вид a^n, $a > 1$). Ето защо сега ще опишем метод, който позволява съществено съкращаване броя на стъпките в силови алгоритми от типа "пълно обхождане" на всички възможности.

Основната идея на този метод (англ. backtracking) е следната:

Методът се прилага, когато търсеното решение трябва да бъде последователност от вида $\langle x_1, x_2, ..., x_n \rangle$ (вж. параграф 1.7 "жаден алгоритъм" за оцветяване върховете на граф). Алгоритъмът строи решението, стартирайки от празна последователност. В общия случай при наличието на частично решение

$\langle x_1, \ldots, x_i \rangle$, се правят опити да се намери такова допустимо значение x_{i+1}, с което намереното частично решение да се разшири до $\langle x_1, \ldots, x_i, x_{i+1} \rangle$. Не е ясно обаче, дали $\langle x_1, \ldots, x_{i+1} \rangle$ може да се разшири до друго решение. Ето защо:

1. Ако такова предполагаемо, но още неизползвано значение x_{i+1} съществува, ние го добавяме към частичното решение и продължаваме процеса за последователността $\langle x_1, \ldots, x_i, x_{i+1} \rangle$ (когато и тя се явява частично, а не пълно решение).

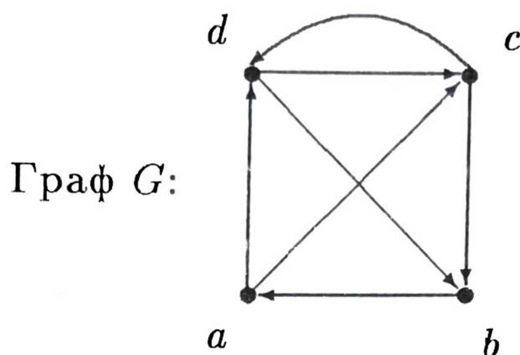
2. Ако такова x_{i+1} не съществува, ние се връщаме към частичното решение $\langle x_1, \ldots, x_{i-1} \rangle$ и продължаваме процеса, търсейки ново, неизползвано допустимо значение x_i'. Сега е ясно, защо този метод се нарича търсене с връщане назад.

Схематично този процес може да се опише така [44]:

```
begin
   k := 1;
   while k > 0 do
      if съществува още неизползван елемент
      y ∈ A_k, такъв че P(X[1], ..., X[k − 1], y) then
         begin X[k] := y; (*елементът y е използван*)
            if ⟨X[1], ..., X[k]⟩ се явява целочислено решение
            then
               write (X[1], ..., X[k]);
            k := k + 1
         end
   else
   (*връщане към по-кратко частично решение; всички елементи на множеството A_k отново стават неизползвани*)
    k := k − 1
end
```

Очевидно е предимството на този метод в сравнение с използвания в края на параграф 1.6 метод за търсене на хамилтонови цикли, който изискваше твърде голям обем памет. В случая се работи с последователност, която нараства до момента на получаване на хамилтонов цикъл или докато не стане ясно, че тази последователност не води до хамилтонов цикъл и последователността се модифицира.

Ще илюстрираме метода търсене с връщане назад. Да разгледаме отново графа от пример 6.5:



Графът G може да бъде представен със списък по следния начин:

	a	b	c	d
1	c	a	b	b
2	d	–	d	c

В стълбовете са описани съседите на всеки връх

Търсенето на всички хамилтонови цикли стартира от връх a.

	Последователност	*Коментар*
1.	a	Добавяме първия възможен връх от стълба на a, т.е. c.
2.	a, c	Добавяме първия възможен връх от стълба на c, т.е. b.
3.	a, c, b	В стълба на b няма възможен връх за добавяне. Backtracking.
4.	a, c	Добавяме върха d.
5.	a, c, d	Добавяме върха b.
6.	a, c, d, b	Хамилтонов цикъл. Backtracking.
7.	a, c, d	Backtracking.
8.	a, c	Backtracking.
9.	a	Добавяме върха d.
10.	a, d	Добавяме върха b.

11.	a, d, b	Backtracking.
12.	a, d	Добавяме върха c.
13.	a, d, c	Добавяме върха b.
14.	a, d, c, b	Хамилтонов цикъл. Backtracking.
15.	a, d, c	Backtracking.
16.	a, d	Backtracking.
17.	a	Backtracking.
18.	$\emptyset$	Край.

ЗАДАЧА 8.2. Да се даде рекурсивен вариант на алгоритъма за търсене с връщане назад (в този случай backtracking-ът не се появява в явен вид).

ЗАДАЧА 8.3. Съставете програма за намиране на всички хамилтонови цикли.

Литература

Minieka, E., *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, Inc., New York and Basel, 1978 (Майника, Э. *Алгоритмы оптимизации на сетях и графах*, М., "Мир", 1981).

Christofides, N., *Graph Theory. An Algorithmic approach*, Academic Press Inc (London) Ltd. 1975, 1977.

Swamy, M., K. Thulasirman., *Graphs, Networks and Algorithms*, John Wiley & Sons, 1981.

Harary, F., *Graph Theory*, Addison-Wesley, Reading, 1969 (Харари, Ф. *Теория графов*, М., "Мир", 1973).

Берж, К., *Теория графов и ее применения*, ИЛ., М., 1962.

Ramamoorthy, C. V., *Analysis of Graphs by Connectivity Considerations*, J. of ACM, 13, p. 211, 1966.

Harary, F, R.Z. Norman, D. Cartwright, *Structural Models: An Introduction to the Theory of Directed Graphs*, Wily, New York, 1965.

Wange, D.L. and D.L.Kleitman, *On the Existence of n-Connected graphs with Prescribed Degrees* ($n \geq 2$), Networks, Vol. 3, 225 - 239, 1973.

Erdos, P. and T. Gallai, *Graphs with Prescribed Degrees of Vertices (in Hungarian)*, Mat. Lapok. Vol. 11, 264-274, 1960.

Tutte, W.T., *The Dissection of EEquilateral Triangles into Equilateral Triangles*, Proc. Cambridge Phil. Soc., Vol. 44, 203-217, 1948.

Welch, J.T., *A Mechanical Analysis of the Cyclic Structure of Indirected Linear Graphs*, Jl. og ACM, 13, p. 205, 1966.

Болобаш, Б., *Теория на графите, Наука и изкуство, София, 1989.*

Чуканов, В., *Комбинаторика, Народна просвета, София, 1977.*

Хаджииванов, Н., *Числа на Рамзи*, Народна просвета, София, 1982.

Сапунджиев, Г., *Може би "да" — може би "не"*. Техника, София, 1978.

Мартинов, Н., К. Петров., *Избрани въпроси по геометрия*, Народна, *просвета, София, 1975.*

Липский, В., *Комбинаторика для программистов*, Мир, Москва, 1988.

Wang, D.L., *A Note on n-Edge Connectivity*, SIAM, J. Appl. Math. Vol. 26, 313-314, 1974.

Menger, K., *Zur Allgemeinen Kurventheorie*, Fund. Math. 10, 96-115, 1927.

Halmosh, P.R., H.E. Vaughan, *The Marriage Problem*. Am. J. Math., Vol. 72, 214-215, 1950.

Rado, R., *Note on thr Transfinite Case of Hall's Theorem on Representatives*, J. London Math. Soc., Vol. 42, 321-324, 1967.

Kaufmann, A., *Graphs, Dynamic Programming and Finite games*, Academic Press, New York, 1967.

Dirac, G.A., *Some Theorems an Abstract Graphs*, Proc. London Math. Soc., Vol. 2, 69-81, 1952.

Ore, O., *Arc Coverings of Graphs*, Ann. Mat. Pura Appl., Vol. 55, 315-321, 1961.

Posa, L., *A Theorem Concerning Hamilton Lineas*, Maguar Tud. Akad. Mat. Kutato Int. Kozl., Vol. 7, 225-226, 1962.

Bondy, J.A., *Properties of Graphs with Constraints on Degrees*, Studia Sci. Math. Hungar., Vol. 4, 473-475, 1969.

Harary, F., Nash-Wiliams C. S. J. A., *On Eulerian and Hamiltonian Graphs and Line Graphs*, Canadian Math, Bulletin, 8, p. 701, 1965.

[28] Бъчваров, Ст. и колектив (под редакцията на Ат. Раденски), *Задачи по програмиране с решения на Паскал*, Унив. изд. "Кл. Охридски", С., 1989.

[29] Whitney, H., *Non-Separable and Planar Graphs*, Trans. Am. Math. Soc., Vol. 34, 339-362, 1932.

[30] Tutte, W. T., *How to Drow a Graph*, Proc. London Math. Soc., Vol. 18, 743-767, 1963.

[31] Wagner, K., *Uber eine Eigenschaft der ebenen Komplexe*, Math. Ann., Vol. 114, 570-590, 1937.

[32] Harary, F. and W. T. Tutte, *A Dual Form of Kuratowski's Theorem*, Canad. Math. Bull., Vol. 8, 17-20, 1965.

[33] MacLane, S., *A Structural Characterization on Planar Combinatorial Graphs*, Duke Math. J., Vol. 3, 340-372, 1937.

[34] Whitney, H., *Planar Graphs*, Fund. Math., Vol. 21, 73-84, 1933.

[35] Parsons, T. D., *On Planar Graphs*, Am. Math. Monthly, Vol. 78, 176-172, 1971.

[36] Appel, K. I. and W. Haken, *Every Planar Map is Four-Colourable*, Bull. Am. Math. Soc., Vol. 82, 711-712, 1976.

[37] Ore, O., *The Four-Color Problem*, Academic Press, New York, 1967.

[38] Saaty, T. L., *Thirteen Colorful Variations on Guthrie's Four-Colour Conjecture*, Am. Math. Mounthly, Vol. 79, 2-43, 1972.

[39] Saaty, T. L. and P. C. Kainen, *The Four-Colour Problem: Assaults and Conguests*, McGraw-Hill, New York, 1977.

[40] Whitney, H. and W. T. Tutte, *Kempe Chains and the Four Colour Problem, in Studies in Graph Theory*, Part *II*, MAA Press, pp. 378-413, 1975.

[41] Welsh D. J. A., M. B. Powell, *An Upper Bound on the Chromatic Number of a Graph and its Application to Timetabling Problems*, The Computer J., 10, p. 85, 1967.

[42] Wood, D. C., *A Technique for Colouring a Graph Applicable to Large Scale Timetabling Problems*, The Computer J., 12, p. 317, 1969.

[43] Williams, M. R., *A Graph Theory Model for the Solution of Timetables*, Ph. D. Thesis, University of Glasgow, 1968.

[44] Brooks, R. L., *On Colouring the Nodes of a Network*, Proc. Cambridge Phil. Soc., Vol. 37, 194-197, 1941.

[45] Melnikov, L. S. and V. G. Vizing, *New Proof of Brooks' Theorem*, J. Combinatorial Theory, Vol. 7, 289-290, 1969.

[46] Heawood, P. J., *Map Colour Theorems*, Quart. J. Math., Vol. 24, 332-338, 1890.

[47] Vizing, V, G., *On an Estimate of the Chromatic Class of a p-Graph* (in Russian), Diskret. Analiz., Vol. 3, 25-30, 1964.

[48] Best M.R., van Emde Boas P., Lenstra H. W., *A sharpened version of the Aandreaa - Rosenburg conjecture*. Techn. Rep. ZW 30/74, Mathematisch Centrum, Amsterdam, 1974.

[49] Rivest R. L., Vnillemin J., *A Generalization and Proof of the Aandreaa - Rosenburg conjecture.*, Proc. 7th Annual ACM Symp. on theory of Computing, May 5-7, 1975, Albuquerque, NM, s. 6-11.

[50] Tarjan R. E., *Depth First Search and Linear Graph Algorithms.* SIAM J. Comput., s. 146-160, 1972.

[51] Danielson G. H., *On Finding the Simple paths and Circuits in a Graph,* IEEE Trans., CT-15, p. 294, 1968.

[52] Dhawan V., *Hamiltonian Circuits and Related Problems in Graph Theory,* M. Sc. Report, Imperial College, London, 1969.

[53] Yan S. S., *Generation of All Hamiltonian Circuits, Paths and Centers of a Graph and Related Problems*, IEEE Trans., CT-14, p. 79, 1967.

Глава 2

Оптимизационни алгоритми в графи и мрежи

2.1. Оптимизационни задачи. Линейно оптимиране

Съществуването на големи, сложни макросистеми налага необходимостта от създаване на обща теория за тяхното управление. Засега решаването на този проблем се ограничава в областта на изследване на отделни класове от такива системи или техни подсистеми — промишлени предприятия, отрасли, икономика, военен комплекс и т.н.

Математическото оптимиране има за предмет теоретичното изследване и разработка на ефективни числени методи за решаване на класове екстремални задачи, свързани с оптимизация при вземането на решения в икономиката, техниката и други сфери. С други думи, това е наука, занимаваща се с разработка и практическо приложение на методи за ефективно управление на системите.

Най-общо понятието *система* може да дефинираме като голяма група от елементи (хора, предприятия, средства за производство и др.), които си взаимодействат непрекъснато. Освен това системата се намира в някакво пространство (всичко останало, невключено в нея), което обикновено се нарича *околна среда на системата*. Очевидно е, че системата като цяло влияе на околната среда и обратно, околната среда я променя. Най-често системата се намира в множество състояния (*динамичен случай*) или съществува само в едно състояние (*статичен случай*). Поради непрекъснатото взаимодействие на нейните части (определени по някакъв признак), тя не може да бъде изучавана без *системен подход* — решаването на произволна задача, колкото и частна да изглежда тя на пръв поглед, влияе върху функционирането на цялата система. При това е необходимо да се осъществява непрекъсната приемственост при прехода от една задача към друга, за да се получи реален ефект. Непрекъснато трябва да се анализира и описва състоянието на системата, както и да се прогнозира състоянието ѝ в бъдеще.

Това налага системата да бъде описана с определени средства, т.е. да се създаде модел, който я отразява, като между модела и системата трябва да има адекватно съответствие.

1. Моделиране

Моделът е някакъв материален или абстрактен обект, който се намира в определено обективно съответствие с изследвания реален обект и носи определена информация за него.

В зависимост от степента на съответствие между математическия модел и оригинала, моделите се делят на *изоморфни* и *хомоморфни*.

Изоморфните модели строго съответстват на оригинала и дават за него пълна информация — нещо, което може да се направи само за прости системи.

Реално съществуващите системи е невъзможно да бъдат описани напълно от модела — за целта се правят редица опростяващи предположения. С други думи, хомоморфните модели са такива модели, които отразяват само някои определени свойства на реалния проблем, т.е. те са значително по-прости от обекта, което прави възможен техния анализ и изследване. Съществуват и други принципи за класификация на математическите модели.

При разработката на математическия модел задължително се спазват следните принципи:

- изучаване и анализ на причинно-следствените връзки;
- използване на аналогии със сходни системи, имащи по-проста структура;
- провеждане на експерименти до изясняване на съществените показатели за работа на системата.

По този начин, след като са изяснени съществените фактори, причинно-следствените връзки, т.е. получен е достатъчно строг и логически непротиворечив, съдържателен модел (*формализация на задачата*), трябва да се премине към намиране на оптимално решение. Това означава да се намерят стойности на участващите в модела променливи

$$x_1,\ x_2, \ldots,\ x_n \quad \text{(може и да не са краен брой)},$$

така че да са изпълнени съотношенията, съществуващи в организацията на системата. Най-често това са някакви ограничения от тип равенство или неравенство —

$$f(x_1, \ldots, x_n) = 0, \quad h(x_1, \ldots, x_n) \leq 0, \quad g(x_1, \ldots, x_n) \geq 0.$$

Броят на тези ограничения може да бъде произволно голям, но краен. Освен казаното дотук, намерените стойности на променливите трябва да бъдат такива, че функцията

$$L(x_1, ..., x_n),$$

която отразява степента на доближаване до поставената цел (*целева функция*), да достига оптимална стойност. Най-често се търси максимумът или минимумът на тази функция.

Векторът $X = (x_1, \ ..., \ x_n)$, удовлетворяващ ограниченията на задачата, се нарича *план*, а векторът X^*, за който $L(X^*)$ е *max* или *min*, се нарича *решение (оптимален план)*.

Променливите, участващи в модела, могат да бъдат: *статични* (независещи от времето) или *динамични; дискретни* (зададени само в отделни моменти от време) или *непрекъснати; детерминирани* или *случайни* (приемат различни стойности с определена вероятност).

Величините, описващи околната среда, се наричат *параметри* (константи).

В общия случай математическият модел има вида

$$L(X, Y) \to \max\ (\min) \tag{1.1}$$

при ограничения (условия)

$$g_i(X, Y) \leq b_i, \ i = 1, \ 2, ..., \ m,$$

където $L(X, Y)$ е целевата функция (показател за качество или ефективност), X е векторът на променливите, а Y е векторът на неуправляемите променливи. Функцията g_i обикновено се нарича функция на потребление на i-тия ресурс, а b_i е величината на този ресурс.

В зависимост от вида и структурата на целевата функция, и ограничаващите условия говорим за:

а) *линейно оптимиране* — целевата функция и ограниченията са линейни относно променливите x_i;

б) *нелинейно оптимиране* — целевата функция или ограниченията са нелинейни относно променливите x_i;

в) *динамично оптимиране* — целевата функция $L(X)$ е адитивна или мултипликативна функция, т.е. има някаква специална структура

$$L(x_1, ..., x_n) = \sum_i L_i(x_i) \ \text{ – адитивна функция,}$$

$$L(x_1, ..., x_n) = \prod_i L_i(x_i) \;\; - \;\; \text{мултипликативна функция;}$$

г) *стохастично оптимиране* — когато векторът Y на неуправляемите променливи е случаен;

д) *дискретно оптимиране* — на променливите x_i е наложено условие за дискретност, например целочисленост. По-общо в дискретното програмиране $X = (x_1, x_2, ..., x_n) \in D$, където D е крайно или изброимо множество. Това условие за дискретност често се разделя по променливите, т.е. $x_j \in D_j,\ j = 1, 2, ..., n$. В този смисъл целочисленото програмиране се явява частен случай на дискретното;

е) *евристично оптимиране* — когато е невъзможно намирането на точно алгоритмично решение, поради огромния брой варианти. В тези случаи вместо оптимално, се търси достатъчно добро, удовлетворително за практиката решение.

Най-добре е изследвано и изучено линейното оптимиране.

В реалните оптимизационни задачи често се търси решение, при наличието на няколко целеви функции $L_1(X), L_2(X), ..., L_m(X)$ отразяващи различни аспекти на проблема.

Намирането на решение X^* в множеството от допустими решения $\{X\}$, което минимизира едновременно всички целеви функции L_i, е възможно, но в изключително благоприятни случаи, рядко срещани в практиката. Ето защо при решаване на *многокритериални, многоцелеви* задачи, които изискват постигането на противоречащи си цели (максимална печалба и вложения за социално осигуряване например), се търси *компромисно решение.* Най-напред се изяснява *степента (теглото) на важност* на всяка от целите — прави се от специалисти в съответната област, след което се оценява *степента на достигане на целите.* Очевидно две или повече противоречащи си цели могат да се осъществяват само частично. Много често степените на важност на целите са сравними помежду си, т.е. смислено е да се въведе, да се говори за йерархия на важност в множеството от всички цели. В други случаи е съдържателно (налага се) целите да се групират по *ниво на важност*, като целите от дадено ниво са сравними помежду си, но никои две цели от различни нива не могат да се сравняват, т.е. имаме йерархия на нивата. В тези случаи се преследва постигането на целите от възможно най-високото ниво, а след това (при възможност) се търси осъществяване на целите от по-ниско ниво на важност.

2. Типични класове оптимизационни задачи

1. ЗАДАЧИ ЗА УПРАВЛЕНИЕ НА ЗАПАСИТЕ: При тези задачи се търси ниво на запасите в складове, при което е минимална сумата на очакваните разходи за съхранението на запасите и загубите от техния дефицит. Да поясним, че при тези задачи големите запаси изискват и големи загуби от съхранение, но намаляват загубите от възможния дефицит. В зависимост от условията, задачата за управление на запасите има различни модификации.

а) Фиксирани са моментите $\{t_1, t_2, ..., t_n\}$ на поръчките за доставка. Да се определят обемите на поръчаните стоки $\{y_1, y_2, ..., y_n\}$.

б) Обемите на стоките $\{y_1, y_2, ..., y_n\}$ са фиксирани. Да се определят моментите на доставка $\{t_1, t_2, ..., t_n\}$.

в) Както моментите на доставка $\{t_1, t_2, ..., t_n\}$, така и обемите $\{y_1, y_2, ..., y_n\}$ не са зададени. Да се определят тези величини, за да се минимизира приетия критерий за оптималност.

2. ЗАДАЧИ ЗА РАЗПРЕДЕЛЕНИЕ НА РЕСУРСИ: Една възможна такава задача е следната. Известни са наличните ресурси (машини, машинно време, брой цехове и т.н.). Да се определи асортиментът (номенклатурата) на възможните производства (работи) така, че отчитайки наличните ресурси, да се обезпечи максимална печалба.

Възможна е и "обратната задача". Зададени са някакви операции, работи, действия, които трябва да се извършат. Да се определи какви ресурси са необходими, с цел да се минимизират сумарните разходи за производство. Например, известно е месечното разписание за движение на самолетите по авиолиниите. Да се определи какъв брой екипажи е необходим, така че планът за превозите да се изпълни с минимални експлоатационни разходи.

3. ЗАДАЧИ ЗА РЕМОНТ И ПОДМЯНА НА ОБОРУДВАНЕ: С течение на времето всяко оборудване се амортизира. Остаряващото оборудване в някакви моменти от време може да се ремонтира или напълно да бъде подменено. И в двата случая

се налагат разходи, свързани с ремонта или подмяната, и загуби, идващи от престоя на машините. Най-общо проблемът е как да се организира този процес във времето, така че да се минимизират средните загуби от ремонт и подмяна на това оборудване за периода на неговия жизнен цикъл.

4. ЗАДАЧИ ЗА МАСОВО ОБСЛУЖВАНЕ: Най-общо това са производствени или битови задачи, при които се образуват опашки, т.е. потокът от заявки за обслужване е неуправляем и случаен, а броят на обслужващите обекти (машини, летища, служби, комуникации и т.н.) е ограничен. Ясно е, че при този тип задачи големият брой обекти за обслужване ще премахне опашките (което би имало положителен производствен ефект например), но за сметка на това ще се получат загуби от дълги престои на обслужващите обекти. Обратно, малкият брой обслужващи обекти ще води до значителни опашки, а оттам и до загуби от чакането в опашка. От казаното е ясно, че една формулировка на този тип задачи е следната: Да се определи броят на обслужващите обекти, при който се минимизира сумата на очакваните загуби (от несвоевременно обслужване и от престой на обектите).

5. ЗАДАЧИ ЗА МРЕЖОВО ПЛАНИРАНЕ И УПРАВЛЕНИЕ: Често се налага разработката и изпълнението на сложни и скъпоструващи проекти, представляващи съвкупност от отделни операции (голям строителен обект например). Освен множеството от операции, е известно и отношението предшестване при изпълнението на тези операции. Най-общо за всяка операция е известно какви операции предшестват нейното изпълнение и кои трябва да се изпълнят след нея. Известна е и взаимната връзка между величината на потребяваните ресурси и продължителността на всяка операция. За целта се съставят мрежови графици — графи, в които дъгите изобразяват операциите на проекта, а върховете са някакви абстрактни събития. Определя се най-дългият път между началото на проекта и събитието, изобразяващо края му. Операциите от този път се наричат *критични* и за тях не са допустими закъснения (закъснението при изпълнение на коя да е от тях води до несвоевременно завършване на целия проект). Определят се най-ранните и най-късните срокове за настъпване на отделните събития. Проблемът е в това, как да се разпределят ресурсите и закъсненията (допустими за некритичните операции), за да се осигури своевременното завършване на целия проект.

6. ЗАДАЧИ ЗА РАЗПОЛОЖЕНИЕ НА ОБЕКТИ: Често се налага за дадена структура допълнително да се разполагат обекти — болници, централи, бензиностанции и т.н. При това, разполагането може да бъде статично или динамично. Въпросът е в това, колко на брой (като се отчитат или не се отчитат досега съществуващите обекти) и къде да бъдат разположени тези обекти, така че да се оптимизира критерият за ефективност. Това може да бъде някаква минимаксна задача — например да се разположи болница така, че разстоянието до най-отдалечената точка в града да бъде минимално. Или пък да се разположи по такъв начин обект, че сумарното разстояние от него до всички структурни единици на региона да бъде минимално.

7. ЗАДАЧИ ЗА ИЗБОР НА МАРШРУТ (МРЕЖОВИ ЗАДАЧИ): Типична такава задача е намирането на маршрут между дадени два града (или между всеки два града), който е най-кратък. В случая, разбира се, са възможни много модификации на тази задача. Друга типична задача от този тип е задачата за намиране на максимален поток. Представете си една мрежа от комуникации (пътища, нефтопроводи и т.н.), в която има начален пункт s и краен пункт t. Известни са пропускателните способности на всяка от комуникациите и цената за пренос на единица от потока (стоки, хора и т.н.) по всяка комуникация. Най-общо проблемът е да се установи какъв е максималният поток между пунктовете s и t, който може да бъде пропуснат в тази мрежа. Или пък как може v единици от даден поток да бъдат пропуснати между пунктовете s и t с минимални разходи.

8. КОМБИНИРАНИ ЗАДАЧИ: Комбинираната задача се състои от няколко отделни задачи и един възможен подход за нейното решаване е следният. Получава се оптимално решение на една от задачите и в зависимост от получения оптимум се намира най-добро решение на следващата задача и т.н. Такъв подход очевидно не винаги води до оптимално решение на комбинираната задача. Но може да се прилага в случаите, когато не съществува метод за търсене на оптимално решение на комбинираната задача като цяло.

Процесът, формализиращ условието на една практическа задача с езика на математиката, се нарича *математическо моделиране* — съставяне на математически модел. Математическият модел най-общо има вида: Да се намери

$$(1.2)\quad \min_{X\in P} L(X) \quad \text{или} \quad \max_{X\in P} L(X), \quad P - \text{ множеството от планове}$$

при ограничения

$$(1.3)\qquad \begin{array}{l} f_i(X) \le b_i,\ i = 1,\ 2, \ldots,\ s, \\ f_i(X) = b_i,\ i = s+1, \ldots,\ n. \end{array}$$

Ще дадем някои основни резултати от линейното оптимиране.

3. Задача на линейното оптимиране

В задачата на линейното оптимиране ограниченията са от тип равенство, от тип неравенство и условия за неотрицателност на променливите. Променливите, за които не е наложено изискването за неотрицателност, ще наричаме *свободни променливи*. Общият вид на задачата на линейното оптимиране е:

Да се намери максимумът (минимумът) на линейната функция

$$(1.4)\qquad L(X) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

при ограничения за променливите ѝ:

$$(1.5)\qquad \left[\begin{array}{lllll} a_{11}x_1 & +a_{12}x_2 & +\ldots+a_{1n}x_n & = b_1 \\ a_{21}x_1 & +a_{22}x_2 & +\ldots+a_{2n}x_n & = b_2 \\ \ldots & \ldots & \ldots & \ldots \\ a_{s1}x_1 & +a_{s2}x_2 & +\ldots+a_{sn}x_n & = b_s \end{array}\right.$$

$$(1.6)\qquad \left[\begin{array}{llll} a_{s+1,1}x_1 & +a_{s+1,2}x_2 & +\ldots+a_{s+1,n}x_n & \le b_{s+1} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1}x_1 & +a_{m2}x_2 & +\ldots+a_{mn}x_n & \le b_m \end{array}\right.$$

$$(1.7)\qquad x_j \ge 0,\ j = 1,\ 2, \ldots,\ r\ (r \le n)$$

При $s = m$ и $r = n$ всички ограничения са от тип равенство и типа (1.7), като условията за неотрицателност се отнасят за

всички променливи. Такава задача на линейното оптимиране се нарича *канонична задача (форма, запис)* .

$$L(X)=\sum_{j=1}^{n}c_jx_j\to\max\ (\min)$$ при условия $$\sum_{j=1}^{n}a_{ij}x_j=b_i,\ i=1,\ 2,...,\ m$$ $$x_j\geq 0,\ j=1,\ 2,...,\ n$$	**Канонична задача**

Когато в общата задача на линейното оптимиране $s=0$ и $r=n$ (т.е. няма ограничения от типа (1.5) и няма свободни променливи), задачата се нарича *стандартна задача*.

$$L(X)=\sum_{j=1}^{n}c_jx_j\to\max\ (\min)$$ при условия $$\sum_{j=1}^{n}a_{ij}x_j\leq b_i,\ i=1,\ 2,...,\ m$$ $$x_j\geq 0,\ j=1,\ 2,...,\ n$$	**Стандартна задача**

В много задачи няма ограничения за неотрицателност на променливите. Различните типове ограничения на линейните оптимизационни задачи обаче не предполагат различни методи за тяхното решаване, защото всички те лесно се свеждат до еквивалентни на тях задачи, записани в канонична форма.

1. Всички ограничения от вида (1.6), т.е. "$\leq$", се преобразу-

ват в равенство чрез прибавяне към лявата част на неравенството на едно ново неотрицателно неизвестно (допълнително неизвестно).

2. Всички неравенства от вида "$\geq$" (без условията за неотрицателност) се преобразуват в равенства, чрез изваждане от лявата им част на едно допълнително неотрицателно неизвестно.

3. Всяко свободно неизвестно x_j, т.е. такова, на което не са наложени ограничения за неотрицателност, може да се представи като

$$x_j = x_j' - \varepsilon,$$

където $x_j' \geq 0$, $\varepsilon \geq 0$ — ε е едно и също за всички свободни неизвестни.

ЗАДАЧА 1.1. Да се намери каноничната форма на задачата

$$L(X) = 3x_1 + 2x_2 - x_3$$

при ограничения

$$\begin{array}{rrrcr} x_1 & -3x_2 & +2x_3 & \leq & -1, \\ 4x_1 & & -2x_3 & = & 5, \\ & 2x_2 & -3x_3 & \geq & 6, \\ x_1 \geq & 0, & x_2 & \geq & 0. \end{array}$$

Решение:

$$\begin{array}{rrrrrcr} x_1 & -3x_2 & +2x_3 & +x_4 & & = & -1, \\ 4x_1 & & -2x_3 & & & = & 5, \\ & 2x_2 & -3x_3 & & -x_5 & = & 6, \end{array}$$

$$x_j \geq 0,\ j = 1,2,4,5,\ x_3 = x_3' - \varepsilon,\ \varepsilon \geq 0,\ x_3' \geq 0.$$

Окончателно получаваме каноничната задача (форма)

$$L(X) = 3x_1 + 2x_2 - x_3' + \varepsilon$$

при ограничения

$$\begin{array}{rrrrrcr} x_1 & -3x_2 & +2x_3' - 2\varepsilon & +x_4 & & = & -1, \\ 4x_1 & & -2x_3' + 2\varepsilon & & & = & 5, \\ & 2x_2 & -3x_3' + 3\varepsilon & & -x_5 & = & 6, \end{array}$$

$$x_j \geq 0,\ j = 1,2,4,5,\ x_3' \geq 0,\ \varepsilon \geq 0.$$

ЗАДАЧА 1.2. Да се определи броят на неизвестните в получената канонична форма, ако изходната задача на линейното оптимиране е неканонична, има n неизвестни x_i, от които s на брой са свободни и p на брой от ограниченията са неравенства.

Отг. $n + p + 1$.

ЗАДАЧА 1.3. Да се докаже, че ако една задача на линейното оптимиране има оптимален план, то и съответната ѝ канонична задача има оптимален план.

	ЗАДАЧА 1.4. Ако $\bar{X}^* = (\bar{x}_1, ..., \bar{x}_p, \bar{x}'_{p+1}, ..., \bar{x}'_n, \bar{x}_{n+1}, ..., \bar{x}_{n+k}, \varepsilon)$ е решение на каноничната задача, запишете решението на изходната задача. *Отг.* $\overline{X} = (\bar{x}_1, ..., \bar{x}_p, \bar{x}'_{p+1} - \varepsilon, \bar{x}'_{p+2} - \varepsilon, ..., \bar{x}'_n - \varepsilon)$.

Като вземем предвид, че коефициентите в левите части на ограниченията (от типа "$\leq$") са обикновено количества от някакъв ресурс за производство на единица продукция, а десните части на ограниченията са наличните количества ресурси, можем икономически да интерпретираме допълнителните неизвестни така (нищо, че не участват в решението на изходната задача):

	1. Ако $\bar{x}_{n+i} = 0,\ i = 1,\ 2, ...$, съответният ресурс b_i ще бъде изчерпан напълно при намерения план. В този смисъл ресурсът b_i може да се счита за дефицитен.
	2. Ако $\bar{x}_{n+i} > 0, i = 1,\ 2, ...$ и това реално количество е прибавено в i-тото ограничение, това означава, че при намерения план ресурсът b_i няма да бъде изразходван напълно с $\bar{x}_{n+i}$ единици.

При неравенства от вида "$\geq$" десните части на ограничението изразяват обикновено някакъв минимален общ ресурс (обем на производство или печалба), а коефициентите в левите страни — ресурсите за единица продукция. Икономическа интерпретация на допълнителните променливи в този случай е:

1. Ако $\bar{x}_{n+i} = 0,\ i = 1,\ 2, \dots$ — съответният ресурс b_i (обем на производството или печалба) ще бъде постигнат. Този оптимален план гарантира изпълнението само на програмата "минимум".

---

2. Ако $\bar{x}_{n+i} > 0,\ i = 1,\ 2, \dots$ и това реално количество е изваждано от i-тото ограничение, това означава, че при намерения план ресурсът b_i (обем производство, печалба) е надхвърлен с $\bar{x}_{n+i}$.

Ако $\overline{X} \in P$ е оптимален план на задачата за намиране на максимум, очевидно следва верността на следните релации

$$\begin{aligned}
&L(\overline{X}) = \max L(X) \Longrightarrow L(X) \leq L(\overline{X}),\ \forall X \\
&\Longrightarrow -L(X) \geq -L(\overline{X}),\ \forall X \Longrightarrow \\
&\Longrightarrow \min(-L(X)) = -L(\overline{X}), \\
&\Longrightarrow -\min(-L(X)) = L(\overline{X}) \Longrightarrow \\
&\Longrightarrow \max_{X \in P} L(X) = -\min_{X \in P}(-L(X)).
\end{aligned}$$

Напълно аналогични разсъждения, когато $\overline{X} \in P$ е план на задачата за търсене на минимум и гореказаното, са достатъчно основание да твърдим, че задачите $\max L(X)$ и $\min L(X)$ имат едно и също множество от планове P. Нещо повече:

За произволна функция $L(X)$ и произволно множество от планове P

$$\max_{X \in P} L(X) = -\min_{X \in P}(-L(X)),$$

$$\min_{X \in P} L(X) = -\max_{X \in P}(-L(X)).$$

4. Канонична задача. Базис на опорен план

Както отбелязахме, всяка задача на линейното оптимиране може да се сведе до еквивалентна на нея канонична задача.

(1.8) $L(X) = (C, X) \to \max$ при ограничения (1.9) $A_1x_1 + A_2x_2 + ... + A_nx_n = B$, (1.10) $X \geq 0$.	**Векторен запис на каноничната задача**

$A = (a_{ij})_{m,n}$ — *матрица на ограниченията* (условията);

B — *вектор на ограниченията*;

A_j — *вектор на условията*, съответстващ на неизвестното x_j, $j = 1, 2, ..., n$;

$X = (x_1, x_2, ..., x_n)$ — *вектор на неизвестните*.

Множеството от планове на задача (1.8) — (1.10) се състои от всички вектори на неизвестните, чиито координати удовлетворяват (1.9) и (1.10).

Ако рангът на матрицата A е различен от ранга на разширената матрица на системата (1.9), от линейната алгебра знаем, че системата (1.9) няма решение, т.е. множеството от планове на задачата ще бъде празното множество.

Интерес за оптимирането представлява случаят, когато системата (1.9) има решение. Без ограничение на общността можем да считаме, че

$$rank(A) = m \leq n.$$

Това е така, тъй като $rank(A) \leq \min(m, n)$ и освен това, ако $rank(A) < m$, след изключване на съответен брой уравнения от системата (1.9) ще достигнем до система, еквивалентна на дадената, с толкова уравнения, колкото е рангът на A.

ОПРЕДЕЛЕНИЕ 1.1. Един план на каноничната задача е **опорен**, *ако векторите на условията, съответстващи на ненулевите му компоненти, са линейно независими.*

Ако нулевият вектор е план на задачата, той е опорен план. Всеки план с единствена строго положителна компонента също е опорен.

Опорните планове имат не повече от m ненулеви компоненти ($m = rank(A)$).

Един план се нарича *изроден*, ако броят на ненулевите му компоненти е по-малък от m. Всяка задача на линейното оптимиране, която има изроден опорен план, ще наричаме *изродена задача*.

ЗАДАЧА 1.5. Дадени са ограниченията

$$\begin{aligned} x_1 + x_2 + x_3 &= 5, \\ 2x_1 + x_2 + 3x_3 &= 9. \end{aligned}$$

Да се определи кои от следните вектори са опорни планове за тези ограничения:

$$X^1 = (4, 1, 0), \qquad X^3 = (2, 2, 1),$$

$$X^2 = (0, 3, 2), \qquad X^4 = \left(1, \frac{5}{2}, \frac{3}{2}\right).$$

Решение: Всеки от векторите X^1, X^2, X^3 и X^4 е план, тъй като компонентите им удовлетворяват ограниченията (директна проверка).

Векторите на условията, съответстващи на ненулевите компоненти на плана X^1, $A_1 = \binom{1}{2}$ и $A_2 = \binom{1}{1}$ са линейно независими, тъй като

$$\begin{vmatrix} 1 & 1 \\ 2 & 1 \end{vmatrix} = -1 \neq 0.$$

Следователно планът X^1 е опорен. Аналогично X^2 също е опорен.

Плановете X^3 и X^4 не са опорни, защото броят на ненулевите им компоненти е по-голям от m $(3 > 2)$.

Опорните планове X^1 и X^2 са неизродени.

ЗАДАЧА 1.6. Има ли изродени опорни планове за ограниченията от задача 1.5? Защо?

ЗАДАЧА 1.7. Дадени са ограниченията

$$\begin{aligned} 5x_1 - x_2 - x_3 &= 5, \\ x_1 + x_2 - x_3 &= 1. \end{aligned}$$

Да се определи дали векторите $X^1 = (1, 0, 0)$, $X^2 = (2, 2, 3)$ са опорни планове за ограниченията.

Отг.: X^1 е опорен и изроден; X^2 не е опорен.

ОПРЕДЕЛЕНИЕ 1.2. **Базис на опорен план** *е всяка съвкупност от m неизвестни x_j, $j = 1, 2, \ldots, n$, чиито съответни вектори на условията са линейно независими и която съдържа всички строго положителни неизвестни x_j от опорния план.*

Ясно е, че един неизроден опорен план има точно един базис — променливите, съответстващи на ненулевите компоненти на плана. Ако опорният план обаче е изроден (броят на ненулевите компоненти е по-малък от m), тогава съвкупността от променливите, съответстващи на тези компоненти, може да се допълни до базис по различни начини.

Планът $X^1 = (1, 0, 0)$ от задача 1.7 е изроден. Всеки негов базис се състои от две неизвестни, тъй като рангът на матрицата на ограниченията е две. Освен това в базиса задължително участва променливата x_1.

Тъй като

$$\Delta_1 = |A_1, A_2| = \begin{vmatrix} 5 & -1 \\ 1 & 1 \end{vmatrix} \neq 0 \quad \text{и} \quad \Delta_2 = |A_1, A_3| = \begin{vmatrix} 5 & -1 \\ 1 & -1 \end{vmatrix} \neq 0,$$

(т.е. векторите на условията са линейно независими), всяка от променливите x_2 и x_3 може да бъде включена в базиса. Следователно базисните представяния на опорния план X^1 са (x_1, x_2) и (x_1, x_3).

ЗАДАЧА 1.8. В канонична задача с размери $m \times n$ е намерен опорен план с k, $k \leq m$ ненулеви компоненти. Какъв е максималният брой базиси на този опорен план?

Отг. C_{n-k}^{m-k}.

Променливите от базиса се наричат *базисни променливи* . Компонентите, съответстващи на тези променливи в плана X, се наричат базисни компоненти, а съответните вектори на условията — *базисни вектори* .

Ограниченията на каноничната задача представляват една линейна система от m уравнения с n неизвестни. При това да припомним, че правим разглежданията при $rank(A) = m$.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m. \end{aligned} \tag{1.11}$$

С метода на Гаус — Жордан можем да решим тази система спрямо m на брой променливи, на които съответните вектори на условията са линейно независими. Без ограничение на общността може да считаме, че това са първите m променливи. Ще получим системата

$$
(1.12)\qquad
\begin{array}{llll}
x_1 & & & +\alpha_{1m+1}x_{m+1} + \dots + \alpha_{1n}x_n = \beta_1 \\
 & x_2 & & +\alpha_{2m+1}x_{m+1} + \dots + \alpha_{2n}x_n = \beta_2 \\
 & & \ddots & \dots\dots\dots\dots\dots\dots\dots\dots\dots \\
 & & x_m & +\alpha_{mm+1}x_{m+1} + \dots + \alpha_{mn}x_n = \beta_m.
\end{array}
$$

Ясно е, че броят на всички такива представяния е краен и е равен на броя на всички m-торки линейно независими вектори-стълбове на условията.

Когато получените свободни членове β_i, $i = 1, 2, \dots, m$ са неотрицателни, т.е. $\beta_i \geq 0$, системата (1.12) се нарича *базисен вид* на системата (1.11) (*базисно представяне*) .

ПРИМЕР 1.1. Да се намерят базисните представяния на системата

$$
\left|\begin{array}{rrrrl}
x_1 & -x_2 & +x_3 & -x_4 & = 5, \\
-x_1 & +2x_2 & -x_3 & & = 0.
\end{array}\right.
$$

$$
A_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix},\ A_2 = \begin{pmatrix} -1 \\ 2 \end{pmatrix},\ A_3 = \begin{pmatrix} 1 \\ -1 \end{pmatrix},\ A_4 = \begin{pmatrix} -1 \\ 0 \end{pmatrix},\ rank(A) = 2.
$$

Линейно независими са двойките вектори A_1, A_2; A_1, A_4; A_2, A_3; A_2, A_4; A_3, A_4, т.е. можем да имаме най-много 5 базисни представяния. Решаваме системата спрямо x_1 и x_2; x_1 и x_4; x_2 и x_3; x_2 и x_4; x_3 и x_4. Очевидно изходната система има две базисни ппредставяния — спрямо (x_1, x_2) и (x_2, x_3)

$$
\left|\begin{array}{rrrrl}
x_1 & & +x_3 & -2x_4 & = 10, \\
 & +x_2 & & -x_4 & = 5.
\end{array}\right.
$$

(Представянето е базисно, тъй като 5 и 10 са неотрицателни числа.)

Да се върнем отново на базисното представяне (1.12). Ако заместим в това базисно представяне небазисните променливи с нули, т.е.

$$x_{m+1} = x_{m+2} = \dots = x_n = 0,$$

базисните променливи $x_1, \dots, x_m$ ще получат съответно стойности $\beta_1, \beta_2, \dots, \beta_m$.

Тогава векторът

$$\overline{X} = (\beta_1, \beta_2, \dots, \beta_m, 0, \dots, 0)$$

ще бъде решение както на системата (1.12), така и на еквивалентната ѝ система (1.11). Следователно $\overline{X}$ ще бъде опорен план на ограниченията на каноничната задача. Планът $\overline{X}$ наистина е опорен, защото векторите на условията на ненулевите му компоненти са линейно независими. И така всяко базисно представяне на системата (1.11) определя един опорен план. Вярно е следното твърдение:

▷ *ТЕОРЕМА 1.1. На всяко базисно представяне на системата (1.11) (каноничната задача) съответства един опорен план на задачата.*
Всеки опорен план на каноничната задача е съответен на някакво базисно представяне на системата (1.11). ◁

Ако в базисното представяне на системата коефициентите β_i са строго положителни, то съответният опорен план $\overline{X}$ е неизроден. Ако съществува поне едно $\beta_i = 0$, опорният план $\overline{X}$ е изроден.

Обръщаме внимание, че е възможно на различни базисни представяния на системата да съответстват едни и същи опорни планове на задачата. На всеки изроден опорен план $\overline{X}$ съответства поне едно представяне.

▷ *ТЕОРЕМА 1.2. Опорните планове на каноничната задача са краен брой.* ◁

Верността на това твърдение е очевидно следствие от това, че броят на линейно независимите m-торки от вектори на условията е краен брой (не повече от C_n^m) и теорема 1.1.

Следните важни за линейното оптимиране резултати ще формулираме без доказателства.

▷ *ТЕОРЕМА 1.3. Ако каноничната задача има план, то тя има и опорен план.* ◁

▷ *ТЕОРЕМА 1.4. Ако в множеството от планове целевата функция* $L(X)$ *на каноничната задача за максимум е ограничена отгоре, за всеки план* $\overline{X}$ *съществува опорен план* $\overline{Y}$*, за който*

$$L(\overline{Y}) \geq L(\overline{X}). \tag{1.13}$$

◁

СЛЕДСТВИЕ 1. Ако в непразното множество от планове целевата функция на каноничната задача за максимум е ограничена отгоре, то задачата има опорен оптимален план.

Задачата на линейното оптимиране няма решение само ако:
1. Целевата функция е неограничена отгоре (респективно отдолу при търсене на минимум);
2. Множеството от планове е празно.

5. Симплекс-метод

Резултатите, получени за каноничната задача, дават един начин за намиране на решение само измежду опорните планове на задачата на линейното оптимиране. Тъй като опорните планове са краен брой, можем да намерим всички опорни планове и за решение да вземем онзи, при който целевата функция има максимална (минимална) стойност.

Практически този начин е неприложим — една система с m уравнения трябва да се реши спрямо всички m-торки неизвестни, съответстващи на линейно независимите вектори на усло-

вията. Като се вземе предвид, че броят на тези m-торки е

$$C_n^m = \frac{n!}{m!(n-m)!} \approx \frac{2^n}{\sqrt{\pi.m}},$$

се оказва, че броят на аритметичните операции, необходими за намиране на всички планове, е приблизително

$$2^n.m^2.\sqrt{m}.$$

Практическите задачи са обикновено с размери, по-големи от 30×50 и горният метод води до изчислителни процедури от порядъка на хиляди години, което го прави неизползваем в реално време.

Ето защо се използва един универсален метод от групата на точните методи, т.е. метод, с който могат да се решават всички задачи на линейното оптимиране и ако задачата има решение, с него се достига до оптимално решение или се установява неразрешимостта на задачата.

Симплекс-методът е създаден от Джордж Данциг, който през 1947 г. е формулирал общата задача на линейното оптимиране и през 1949 г. е публикувал симплекс-метода за решаването ѝ.

През 1939 г. Л. В. Канторович е създал метод на разрешаващите множители и го е използвал при решаването на някои класове задачи на линейното оптимиране, като по-късно методът е доразвит и получава названието *метод на обективно обусловените оценки* .

Двата метода са построени на един и същ принцип и се различават само в детайли. През 1956 г. Данциг, Форд, Фалкерсон и др. подробно развиват симплекс-метода.

(1.14)	За да се приложи симплекс-методът: 1. Ограниченията в модела на задачата трябва да бъдат от тип равенство. 2. Матрицата A на решаваната система уравнения трябва да съдържа единична матрица от m-ти ред (базисно представяне).

В редица икономически, управленски и други линейни задачи ограничителните условия не са само от тип равенство или пощо не са изпълнени условията (1.14), поради което техните

модели трябва да се приведат към горе цитираните изисквания на симплекс-метода, след което могат да се решават с него. Налице са следните четири възможности.

Първи случай (канонична форма):

$$L = CX \to \max, \quad AX = B, \quad X \geq 0.$$

При задачи с такъв модел:

1. Ако системата е приведена в базисен вид, условията (1.14) са налице и симплекс-методът е приложим (без преобразувания на модела);

2. В останалите случаи моделът се преобразува, като се добавят *изкуствени променливи* до вида

$$L(X) = c_1x_1 + c_2x_2 + ... + c_nx_n - Mx_{n+1} - ... - Mx_{n+m} \to \max$$

при ограничения

$$\begin{array}{llllll} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n & +x_{n+1} & & & & = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n & & +x_{n+2} & & & = b_2 \\ \dots\dots\dots\dots\dots\dots & & \dots & \ddots & & \dots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n & & & & +x_{n+m} & = b_m \end{array}$$

$$x_j \geq 0, \; j = 1, \; 2, ..., \; n, \; n+1, ..., \; n+m$$

(броят на изкуствените променливи е $\leq m$, те се добавят само в онези уравнения, в които това се налага, за да се стигне до базисно представяне).

В целевата функция изкуствените променливи се включват с коефициент $(-M)$, където M е достатъчно голямо положително число (при задачата за минимум с коефициент $(+M)$).

Втори случай (стандартна форма):

$$L = CX \to \max\,(\min), \quad AX \leq B, \quad X \geq 0.$$

При задача с такъв модел към всяко неравенство прибавяме *допълнителна променлива*, което осигурява наличието на изискванията (1.14) и прави приложим симплекс-метода.

Допълнителните променливи се включват в целевата функция с коефициенти нула:

$$L(X) = c_1x_1 + c_2x_2 + ... + c_nx_n + 0.x_{n+1} + ... + 0.x_{n+m}$$

при ограничения

$$\begin{array}{llllll}
a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n & +x_{n+1} & & & & = b_1 \\
a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n & & +x_{n+2} & & & = b_2 \\
\dots\dots\dots\dots & \dots & & \ddots & & \dots \\
a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n & & & & +x_{n+m} & = b_m
\end{array}$$

$$x_j \geq 0, \ j = 1, \ 2, ..., \ n, \ n+1, ..., \ n+m.$$

Трети случай:

$$L = CX \to \min \ (\max), \quad AX \geq B, \quad X \geq 0.$$

1. От всяко неравенство се изважда по една *допълнителна променлива*, за да получим ограничения от тип равенство. В целевата функция допълнителните променливи се включват с нулеви коефициенти.

2. Към уравненията се прибавя по една *изкуствена променлива* с коефициент единица с цел да се образува очевиден базис. В целевата функция при търсене на минимум изкуствените променливи се добавят с коефициент $(+M)$, а при задача за максимум — с коефициент $(-M)$, където M е достатъчно голямо положително число.

Моделът ще добие вида

$$L(X) = c_1x_1 + c_2x_2 + ... + c_nx_n + 0.x_{n+1} + ... + 0.x_{n+m} +$$

$$+M.x_{n+m+1} + ... + M.x_{n+m+m} \to \min$$

при ограничения

$$\begin{array}{llllll}
a_{11}x_1 + ... + a_{1n}x_n & -x_{n+1} + x_{n+m+1} & & & & = b_1 \\
a_{21}x_1 + ... + a_{2n}x_n & & -x_{n+2} + x_{n+m+2} & & & = b_2 \\
\dots\dots\dots & \dots & & \ddots & & \dots \\
a_{m1}x_1 + ... + a_{mn}x_n & & & & -x_{n+m} + x_{n+m+m} & = b_m
\end{array}$$

$$x_j \geq 0,\ j = 1,\ 2, ...,\ n + 2m.$$

Четвърти случай:

$$L = CX,\ AX \lesseqgtr B,\ X \geq 0.$$

Моделът се преобразува, както беше посочено във втори и трети случай.

Ако резюмираме казаното, всички модели на задачата на линейното оптимиране се привеждат към изискванията (1.14) с добавяне (изваждане) на допълнителни променливи или на изкуствени променливи с цел да се образува базис.

Когато базисът на изходното решение се състои само от допълнителни променливи или променливи, съдържащи се в изходната система, симплекс-методът се нарича *симплекс-метод с естествен базис (обикновен симплекс-метод).*

Когато базисът на изходното решение е изкуствено създаден, т.е. съдържа поне една изкуствена променлива, симплекс-методът се нарича *симплекс-метод с изкуствен базис (метод на изкуствения базис, М-метод).*

Решаването на задачата с метода на изкуствения базис ползва алгоритъма на симплекс-метода с тази разлика, че след като една изкуствена променлива се изключи от базиса, не е необходимо да се изчисляват компонентите на нейния вектор-стълб.

Идеята на симплекс-метода е, като се вземе един начален опорен план, да се достигне до друг опорен план, който е по-добър (не по-лош) от предходния. На всяка стъпка се изследва за оптималност опорният план и се прави следваща итерация в случаите, когато това се налага, докато се стигне до опорен план, който е оптимален. Обикновено не се налага намирането на всички опорни планове, за да се стигне до решение на задачата.

Проверка (критерий) за оптималност. Преминаване към друг опорен план

Нека имаме едно базисно представяне на системата на условията

$$
\begin{array}{llll}
x_1 & & & +\alpha_{1m+1}x_{m+1} + \dots + \alpha_{1q}x_q + \dots + \alpha_{1n}x_n = \beta_1 \\
& x_2 & & +\alpha_{2m+1}x_{m+1} + \dots + \alpha_{2q}x_q + \dots + \alpha_{2n}x_n = \beta_2 \\
& & \ddots & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
(1.15) & & x_p & +\alpha_{pm+1}x_{m+1} + \dots + \alpha_{pq}x_q + \dots + \alpha_{pn}x_n = \beta_p \\
& & \ddots & \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
& & & x_m + \alpha_{mm+1}x_{m+1} + \dots + \alpha_{mq}x_q + \dots + \alpha_{mn}x_n = \beta_m,
\end{array}
$$

съответен опорен план $X^0 = (\beta_1, \beta_2, \dots, \beta_m, 0, \dots, 0)$ и решаваме задача за търсене на максимум.

Ако изразим от това базисно представяне базисните неизвестни $x_1, \dots, x_m$ и ги заместим в целевата функция, ще получим

$$L(X) = L_0 - \Delta_{m+1}x_{m+1} - \Delta_{m+2}x_{m+2} - \dots - \Delta_n x_n, \tag{1.16}$$

където

$$\Delta_j = c_1\alpha_{1j} + c_2\alpha_{2j} + \dots + c_m\alpha_{mj}, \; j = m+1, \dots, n,$$

$$L_0 = c_1\beta_1 + c_2\beta_2 + \dots + c_m\beta_m \quad (L_0 \; - \; \text{свободен член}).$$

1. От (1.16) се вижда, че стойността на $L(X)$ може да се увеличи чрез даване на положителни стойности на свободните променливи само ако техните коефициенти Δ_j са отрицателни. Ако всички $\Delta_j \geq 0$, то при даване на положителни стойности на свободните променливи x_j стойността на $L(X)$ не може да се подобри (увеличи). От това следва, че функцията $L(X)$ е достигнала максимума си при плана X^0.

Критерий за оптималност: Един опорен план е оптимален, ако в съответстващия му базисен вид на системата (задачата) всички коефициенти на целевата функция са неотрицателни, т.е.

$$\Delta_j \geq 0, \; j = m+1, \dots, n.$$

2. Нека съществува поне едно $\Delta_j < 0$. Без ограничение на общността да допуснем, че $\Delta_q < 0, \; m+1 \leq q \leq n$.

2а) Нека всички коефициенти

$$\alpha_{1q},\ \alpha_{2q}, ...,\ \alpha_{mq}$$

са неположителни. Да положим в базисното представяне (1.15) $x_q = \theta \geq 0$, а всички останали небазисни променливи да са равни на нула. За базисните променливи ще получим

$$x_i = \beta_i - \alpha_{iq}\theta \geq 0,\ i = 1,\ 2, ...,\ m,$$

защото $\theta \geq 0$ и $\alpha_{iq} \leq 0$. Векторът

$$X(\theta) = (\beta_1 - \alpha_{1q}\theta,\ ...,\ \beta_m - \alpha_{mq}\theta,\ 0, ...,\ 0,\ t,\ 0,\ ...,\ 0)$$

очевидно е план на задачата с ограничения (1.15). Освен това стойността на целевата функция за този вектор $X(\theta)$ е

$$L(X(\theta)) = L_0 - \Delta_q\theta$$

и целевата функция очевидно неограничено ще нараства с нарастването на θ ($\Delta_q < 0,\ \theta \geq 0$).

	Ако в даден базисен вид на каноничната задача целевата функция има отрицателен коефициент Δ_q и всички коефициенти $\alpha_{1q}, ..., \alpha_{mq}$ пред съответната неизвестна x_q са неположителни, то функцията е неограничена отгоре в множеството от планове. Задачата няма решение.

2б) Нека съществува коефициент $\Delta_q < 0$ и за всички такива коефициенти поне един от коефициентите

$$\alpha_{1q},\ \alpha_{2q}, ...,\ \alpha_{mq}$$

е положителен, т.е. $\alpha_{iq} > 0,\ i \in [1, m]$.

В този случай не е изпълнен критерият за оптималност, а не е налице и критерият за неограниченост на целевата функция. В такъв случай се налага да преминем от един опорен план към друг — от едно базисно представяне на системата към друго.

При това преминаване една от базисните променливи на системата става небазисна и на нейно място в базиса се включва

една от небазисните променливи. За намиране на новото базисно представяне трябва да решим едно от уравненията на системата спрямо x_q и да изключим x_q от останалите уравнения.

Освен това трябва да подберем това уравнение така, че след изключване на x_q от останалите уравнения да получим система, която е базисно представяне (за свободните ѝ членове β_i да е изпълнено $\beta_i \geq 0$).

Без ограничение на общността нека изберем p-тото уравнение и с метода на Гаус — Жордан да изключим x_q.

Изискването новата система, която получаваме, да бъде базисно представяне налага

$$\beta'_p = \frac{\beta_p}{\alpha_{pq}} \geq 0 \quad \text{и} \quad \beta'_i = \beta_i - \alpha_{iq}\frac{\beta_p}{\alpha_{pq}} \geq 0, \quad i = 1,\ 2, \dots,\ m, \quad i \neq p.$$

(Тук β'_i са новите свободни членове на системата след преобразуването).

1. За верността на горните изисквания очевидно трябва $\alpha_{pq} > 0$.
2. Горните изисквания са изпълнени за всяко $\alpha_{iq} \leq 0$.
3. За да бъдат споменатите изисквания изпълнени в случаите $\alpha_{iq} > 0$, достатъчно е

$$\frac{\beta_p}{\alpha_{pq}} = \min_{\alpha_{iq}>0} \frac{\beta_i}{\alpha_{iq}}.$$

Уравнението p можем да решим спрямо x_q и да го изключим от останалите уравнения, ако

1. $\alpha_{pq} > 0$ и
2. $\dfrac{\beta_p}{\alpha_{pq}} = \min\limits_{\alpha_{iq}>0} \dfrac{\beta_i}{\alpha_{iq}}$.

Небазисното неизвестно x_q, което влиза в новия базис, се избира измежду онези небазисни неизвестни, за които $\Delta_q < 0$.

По този начин небазисното неизвестно x_q от стария опорен план става базисно неизвестно при новия опорен план, а променливата x_p ще бъде изключена от базиса.

При това преобразуване на системата (нов базис, с нова базисна променлива) за стойността на целевата функция $L(X)$ при новия спорен план X^1 се получава

(1.17)
$$L(X^1) = L_0 - \frac{\beta_p}{\alpha_{pq}} \Delta q \geq L_0, \text{ т.е.}$$

$$L(X^1) \geq L_0,$$

където L_0 е стойността на целевата функция при предишния опорен план.

Новият опорен план X^1, съответен на новото базисно представяне, има вида

$$X^1 = (\beta'_1, \ ..., \ \beta'_{p-1}, \ 0, \ \beta'_{p+1}, \ \beta'_m, \ 0, ..., \ 0, \ \beta'_p, \ 0, \ ..., \ 0).$$

А базисът е $Б(X^1) = (x_1, ..., x_{p-1}, x_q, x_{p+1}, ..., x_m)$.

	Ако опорният план X^0 е неизроден ($\beta_i > 0$), от (1.17) се вижда, че $L(X^1) > L(X^0)$, т.е. целевата функция е подобрила стойността си. Ако всички опорни планове са неизродени, това изключва възможността да се върнем в опорен план, в който вече сме били. Ако опорният план X^0 е изроден, то е възможно $\beta_p = 0$ и $$L(X^1) = L(X^0),$$ т.е. стойността на целевата функция остава същата. Намираме се в същия опорен план, но сме преминали към друг базис.

Когато не е налице критерият за неограниченост на целевата функция и има няколко коефициента $\Delta_j < 0$, от (1.17) се вижда, че най-бързо подобрява целевата функция най-малкият от отрицателните коефициенти Δ_j.

Ще отбележим, че при изродените задачи на линейното оптимиране е теоретично възможно да се получи зацикляне в процеса на тяхното решаване — целевата функция не променя стойността си, оставаме в същия опорен план, сменяйки базиса му и минавайки през различните му базиси, се връщаме в изходния базис. Зациклянето пречи да стигнем до оптималното решение. Практически зацикляне при решаване на икономическо-управленски задачи се получава изключително рядко — само при изродени решения с повече от една базисна нула. Разработени са методи за отстраняване на зациклянето, но те са твърде трудоемки и няма да бъдат предмет на разглежданията в тази книга.

6. Алгоритъм на симплекс-метода. Симплекс-таблици

Да резюмираме получените резултати, описвайки *алгоритъма на симплекс-метода при задача за търсене на максимум.*

1. Намираме базисно представяне (опорен план) на системата условия на задачата (ако няма естествен базис — построяваме изкуствен). Изключваме базисните неизвестни от целевата функция $L(X)$, т.е. намираме коефициентите Δ_j на целевата функция.

2. Ако всички $\Delta_j \geq 0$ — съответният опорен план е оптимален. Край.

3. Съществува $\Delta_j < 0$, за което $\forall \alpha_{ij} \leq 0,\ i = 1,\ 2, ...,\ m$. Целевата функция е неограничена отгоре, т.е. задачата няма решение. Край.

4. За всяко $\Delta_j < 0$, поне един от коефициентите $\alpha_{ij} > 0$, $i = 1,\ 2, ...,\ m$. Избираме ключов стълб q, за който $\Delta_q < 0$.

5. Избираме ключов ред p, за който

$$\frac{\beta_p}{\alpha_{pq}} = \min_{\alpha_{iq}>0} \frac{\beta_i}{\alpha_{iq}}.$$

6. С метода на Гаус — Жордан правим елементарно преобразуване на системата с ключовия елемент α_{pq} и получаваме (нов) опорен план. Връщаме се на 2.

ЗАДАЧА 1.9. Формулирайте алгоритъма на симплекс-метода при задачи за търсене на минимум.

За удобство и прегледност данните на задачата (и междинните резултати от решаването) се записват в *симплекс-таблици*. Най-общо симплекс-таблицата има вида

C	$Б_x$	A_0	c_1	c_2	$\cdots$	c_l	$\cdots$	c_m	c_{m+1}	$\cdots$	c_k	$\cdots$	c_n
			A_1	A_2	$\cdots$	A_l	$\cdots$	A_m	A_{m+1}	$\cdots$	A_k	$\cdots$	A_n
c_1	x_1	β_1	1	0	$\cdots$	0	$\cdots$	0	$\alpha_{1,m+1}$	$\cdots$	α_{1k}	$\cdots$	α_{1n}
c_2	x_2	β_2	0	1	$\cdots$	0	$\cdots$	0	$\alpha_{2,m+1}$	$\cdots$	α_{2k}	$\cdots$	α_{2n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
c_l	x_l	β_l	0	0	$\cdots$	1	$\cdots$	0	$\alpha_{l,m+1}$	$\cdots$	α_{lk}	$\cdots$	α_{ln}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
c_m	x_m	β_m	0	0	$\cdots$	0	$\cdots$	1	$\alpha_{m,m+1}$	$\cdots$	α_{mk}	$\cdots$	α_{mn}
Δ_j		L_0	0	0	$\cdots$	0	$\cdots$	0	Δ_{m+1}	$\cdots$	Δ_k	$\cdots$	Δ_n

Въпреки очевидността, ще дадем следните пояснения:

C — стълб от коефициенти на базисните неизвестни в целевата функция;

$Б_x$ — стълб от базисните променливи (на различните уравнения);

A_0 — стълб от свободните членове на системата;

Първи ред — коефициенти на целевата функция;

Втори ред — променливите на целевата функция (съответните вектори на условията);

Основна част — матрица на системата ограничения;

Последен ред — съдържа стойността L_0 на целевата функция и индексите Δ_j.

$$L_0 = \sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} c_i x_i = \sum_{i=1}^{m} c_i \beta_i, \text{ т.е. } L_0 = (C, A_0).$$

Индексите Δ_j се пресмятат по формулата

$$\Delta_j = (C, A_j) - c_j = \sum_{i=1}^{m} c_i \alpha_{ij} - c_j, \ j = 1,\ 2, \ldots,\ n.$$

Горе с (C, A_0) и (C, A_j) сме означили скаларното произведение на векторите.

Индексите Δ_j на базисните променливи няма нужда да се пресмятат по горната формула. Те са равни на нула.

Казаното ще илюстрираме с примери.

ПРИМЕР 1.2. Да се определи

$$L(X) = 3x_1 - x_2 + 2x_3 \to \max \tag{1.18}$$

при следните ограничения

$$\begin{array}{rrrl} x_1 & +3x_2 & +11x_3 & \leq 11 \\ x_1 & -\ x_2 & +\ 3x_3 & \leq 2 \\ -3x_1 & +\ x_2 & +\ \ x_3 & \leq 3 \end{array} \tag{1.19}$$

$$x_j \geq 0, \ j = 1, \ 2, \ 3.$$

Решение: Както отбелязахме, свеждаме системата неравенства до система уравнения, чрез добавяне на една допълнителна променлива с коефициент единица към всяко от неравенствата. Моделът на задачата ще добие следния удобен за симплекс-метода вид:

$$L(X) = 3x_1 - x_2 + 2x_3 + 0.x_4 + 0.x_5 + 0.x_6 \to \max \tag{1.20}$$

при условия

$$\begin{array}{rrrrrrl} x_1 & +3x_2 & +11x_3 & +x_4 & & & = 11 \\ x_1 & -\ x_2 & +\ 3x_3 & & +x_5 & & = 2 \\ -3x_1 & +\ x_2 & +\ \ x_3 & & & +x_6 & = 3 \end{array} \tag{1.21}$$

$$x_j \geq 0, \ j = 1, \ 2, \ 3, \ 4, \ 5, \ 6.$$

На система (1.21) намираме едно изходно решение (начален опорен план). При $x_1 = x_2 = x_3 = 0$ имаме $x_4 = 11$, $x_5 = 2$, $x_6 = 3$. Променливите x_4, x_5, x_6 са базисни.

Полученият начален план X^0 е

$$X^0 = (0, 0, 0, 11, 2, 3).$$

Построяваме и попълваме симплекс-таблицата.

C	$Б_x$	A_0	3	−1	2	0	0	0
			A_1	A_2	A_3	A_4	A_5	A_6
0	x_4	11	1	3	11	1	0	0
0	x_5	2	[1]	−1	3	0	1	0
0	x_6	3	−3	1	1	0	0	1
Δ_j		$L_0 = 0$	$\Delta_1 = -3$	$\Delta_2 = 1$	$\Delta_3 = -2$	0	0	0

1) $L_0 = (C, A_0) = 0.11 + 0.2 + 0.3 = 0$;

2) $\Delta_4 = \Delta_5 = \Delta_6 = 0$ — променливите x_4, x_5 и x_6 са базисни.

$$3)\quad \begin{aligned} \Delta_1 &= (C, A_1) - c_1 = 0.1 + 0.1 + 0.(-3) - 3 = -3 \\ \Delta_2 &= (C, A_2) - c_2 = 0.3 + 0.(-1) + 0.1 - (-1) = 1 \\ \Delta_3 &= (C, A_3) - c_3 = 0.(11 + 3 + 1) - 2 = -2 \end{aligned}$$

Тъй като не всички $\Delta_j \geq 0$, полученият план не е оптимален. Критерият за неограниченост на целевата функция също не е налице — за $\forall \Delta_j < 0$, $\exists \alpha_{ij} > 0$, $i = 1,\ 2,\ 3$.

Ще преминем към друг опорен план. Ключов стълб може да бъде както A_1, така и A_3 ($\Delta_1 = -3$, $\Delta_3 = -2$). Избираме A_1, тъй като $\Delta_1 = -3$ най-бързо подобрява стойността на целевата функция.

Определяме ключовия ред. Тъй като

$$\theta = \min\left(\frac{11}{1}, \frac{2}{1}\right) = 2,$$

ключов ред ще бъде вторият, т.е. ключовият (разрешаващият) елемент ще бъде елементът α_{21}. Ще извадим от базиса x_5 и на нейно място базисна ще стане променливата x_1.

Прилагаме метода на Гаус — Жордан и преобразуваме таблицата.

Да припомним правилата за преобразуване на матрицата при метода на Гаус — Жордан.

1) Попълваме ключовия ред (старите коефициенти делим на ключовия елемент);
2) Попълваме "новия" базисен стълб (нули в празните клетки);
3) Попълваме оставащите базисни стълбове — тези, които са били и остават базисни;
4) Попълваме останалите празни клетки по правилото

кл.елем. a

c b

$$b_{\text{ново}} = b_{\text{старо}} - \frac{a.c}{\text{кл.ел.}}$$

C	Б$_x$	A_0	3	-1	2	0	0	0
			A_1	A_2	A_3	A_4	A_5	A_6
0	x_4	9	0	4	8	1	-1	0
3	x_1	2	1	-1	3	0	1	0
0	x_6	9	0	-2	10	0	3	1
Δ_j		$L = 6$	0	$\Delta_2 = -2$	$\Delta_3 = 7$	0	$\Delta_5 = 3$	0

Както се очакваше, стойността на целевата функция се подобри с $\Delta.\theta = 3.2 = 6$. Коефициентите Δ_j може да се пресмятат по всеки от двата начина — чрез скаларното произведение на векторите $(C, A_j) - c_j$, или по Гаус — Жордановото правило за изчисляване на елемент от таблицата. Единият от двата начина може да служи за проверка. Да припомним още веднъж и това, че Δ-те на базисните променливи са нули.

За новия план $\Delta_2 = -2 < 0$. Планът не е оптимален. Освен това има коефициент от стълба A_2, който е положителен — това е $\alpha_{12} = 4 > 0$. Преминаваме към друг опорен план с единствено възможния ключов елемент $\alpha_{12} = 4$.

C	$Б_x$	A_0	3	−1	2	0	0	0
			A_1	A_2	A_3	A_4	A_5	A_6
−1	x_2	9/4	0	1	2	1/4	−1/4	0
3	x_1	17/4	1	0	5	1/4	3/4	0
0	x_6	27/2	0	0	14	1/2	5/2	1
Δ_j		$L = 21/2$	0	0	$\Delta_3 = 11$	$\Delta_4 = 1/2$	$\Delta_5 = 5/2$	0

$$L(X) = L\text{старо} - \theta.\Delta = 6 - \left(\frac{9}{4}\right).(-2) = 6 + \frac{9}{2} = \frac{21}{2}$$

$$L(X) = (-1).\frac{9}{4} + 3.\frac{17}{14} + 0.\frac{27}{2} = -\frac{9}{4} + \frac{51}{4} = \frac{42}{4} = \frac{21}{2}$$

$$\Delta_3 = (C, A_3) - c_3 = ((-1).2 + 3.5 + 0.14) - 2 = 11 > 0$$

$$\Delta_4 = (C, A_4) - c_4 = \left((-1).\frac{1}{4} + 3.\frac{1}{4} + 0.\frac{1}{2}\right) - 0 = \frac{1}{2} > 0$$

$$\Delta_5 = (C, A_5) - c_5 = \left((-1).\left(-\frac{1}{4}\right) + 3.\frac{3}{4} + 0.\frac{5}{2}\right) - 0 = \frac{10}{4} = \frac{5}{2} > 0.$$

Разбира се Δ_3, Δ_4 и Δ_5 можехме да пресметнем, като попълним клетките от таблицата с правилото на Гаус — Жордан.

Всички Δ_j на получения план са ≥ 0, следователно този план е оптимален.

$$X\text{опт} = \left(\frac{17}{4}, \frac{9}{4}, 0, 0, 0, \frac{27}{2}\right), \quad L_{\max} = \frac{21}{2}.$$

Забележка: Ако тази задача трябваше да решим като задача за търсене на $\min L(X)$ при същите ограничения:

а) можехме да я превърнем в задача за търсене на максимум, като използваме направените вече теоретични бележки, а именно

$$\min_{X \in P} L(X) = -\max_{X \in P}((-L(X));$$

б) или като променим критерия за оптималност по отношение стойностите на Δ_j. При търсене на минимум, ако $\forall \Delta_{ij} \leq 0$, планът е оптимален.

Ако съществува $\Delta_j > 0$, за което всички $\alpha_{ij} \leq 0$, критерият за неограниченост на целевата функция е налице — задачата няма решение. В останалите случаи минаваме към нов опорен план.

ПРИМЕР 1.3. Да се намери ***минимумът*** на $L(X)$

$$L(X) = -x_1 + x_2$$

при ограниченията

$$\begin{array}{llllll} -2x_1 & +x_2 & +x_3 & & & = 4 \\ x_1 & -4x_2 & & +x_4 & & = 4 \\ x_1 & -2x_2 & & & +x_5 & = 10 \end{array}$$

Решение:

C	$Б_x$	A_0	-1	1	0	0	0
			A_1	A_2	A_3	A_4	A_5
0	x_3	4	-2	1	1	0	0
0	x_4	4	$\boxed{1}$	-4	0	1	0
0	x_5	10	1	-2	0	0	1
Δ_j		$L_0 = 0$	1	-1	0	0	0

Намереният план не е оптимален, $\Delta_1 = 1 > 0$ (решаваме задача за търсене на минимум). Ключов стълб е първият стълб A_1. Тъй като

$$\min\left(\frac{4}{1}, \frac{10}{1}\right) = 4,$$

ключов ред ще бъде вторият, а ключов елемент е α_{21}.

C	$Б_x$	A_0	-1	1	0	0	0
			A_1	A_2	A_3	A_4	A_5
0	x_3	12	0	-7	1	2	0
-1	x_1	4	1	-4	0	1	0
0	x_5	6	0	$\boxed{2}$	0	-1	1
Δ_j		$L = -4$	0	3	0	-1	0

Полученият план не е оптимален. $\Delta_2 = 3 > 0$ и липсва критерият за неограниченост на целевата функция. Преминаваме към нов опорен план с ключов елемент α_{32}.

C	$Б_x$	A_0	-1	1	0	0	0
			A_1	A_2	A_3	A_4	A_5
0	x_3	33	0	0	1	$-3/2$	$7/2$
-1	x_1	16	1	0	0	-1	2
1	x_2	3	0	1	0	$-1/2$	$1/2$
Δ_j		$L = -13$	0	0	0	$1/2$	$-3/2$

Тъй като $\Delta_4 = \frac{1}{2} > 0$ и всички елементи α_{i4} са отрицателни, налице е критерият за неограниченост на целевата функция. Тази целева функция няма най-малка стойност. Задачата няма решение.

7. Симплекс-метод с изкуствен базис (М-задача)

Когато в задачи от типа

(1.22) $$L(X) = (C, X) \to \max$$

при ограничения

(1.23) $$AX = B,\ X \geq 0 \text{ (или } AX \geq B,\ X \geq 0)$$

"липсва" (не се вижда) естествен базис преобразуваме модела на задача (1.22) — (1.23) и получаваме нова разширена задача, наричана още M-задача.

(1.24) $L(X) = c_1x_1 + c_2x_2 + \ldots + c_nx_n - Mx_{n+1} - \ldots - Mx_{n+m} \to \max$ при ограничения (1.25) $\begin{array}{llll} a_{11}x_1 + \ldots + a_{1n}x_n & +x_{n+1} & & = b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n & & +x_{n+2} & = b_2 \\ \ldots & \ldots & \ddots & \ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n & & & +x_{n+m} = b_m \end{array}$ $x_j \geq 0,\ j = 1,\ 2, \ldots,\ n + m.$	**M-задача**

Оптималното решение на задача (1.22) — (1.23), ако има такова, се получава след решаване на M-задачата.

За разширената M-задача се прилагат правилата на симплекс-метода. Разликата се състои в това, че когато една от изкуствените променливи излезе от базиса, съответният на нея вектор-стълб може да се изключи от таблицата. Основание за това дава следната

▷ *ТЕОРЕМА 1.5. Ако в оптималното решение на M-задачата*

$$X_{\text{опт}} = (x_1, x_2, ..., x_n, x_{n+1}, x_{n+2}, ..., x_{n+m})$$

всички изкуствени променливи x_{n+i} са равни на нула, то

$$X^0 = (x_1, x_2, ..., x_n)$$

е оптимално решение на изходната задача (1.22) — (1.23). ◁

Докажете верността на следните две твърдения:

ТВЪРДЕНИЕ 1: Ако оптималното решение на M-задачата съдържа поне една изкуствена положителна променлива, системата (1.23) няма решение.

ТВЪРДЕНИЕ 2: Ако линейната функция (1.24) на M-задачата е неограничена, изходната задача няма решение или поради несъвместимост на системата (1.23), или поради неограниченост отгоре на целевата функция (1.23).

ПРИМЕР 1.4. Да се намери минимумът на функцията

$$L(X) = -4x_1 - x_2 - 3x_3 \to \min$$

при ограничения

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 30 \\ 2x_1 + x_2 + 5x_3 &= 40 \\ x_1 + 2x_2 + x_3 &= 20 \end{aligned}$$

$$x_j \geq 0.$$

Решение: Моделът на M-задачата (разширената) има вида:

$$L(X) = -4x_1 - x_2 - 3x_3 + Mx_4 + Mx_5 + Mx_6 \to \min$$

при ограничения

$$\begin{aligned} x_1 + 2x_2 + 3x_3 + x_4 \qquad\qquad &= 30 \\ 2x_1 + x_2 + 5x_3 \qquad + x_5 \qquad &= 40 \\ x_1 + 2x_2 + x_3 \qquad\qquad + x_6 &= 20 \end{aligned}$$

Попълваме симплекс-таблицата и решаваме M-задачата.

C	$Б_x$	A_0	-4	-1	-3	M	M	M
			A_1	A_2	A_3	A_4	A_5	A_6
M	x_4	30	1	2	3	1	0	0
M	x_5	40	2	1	**5**	0	1	0
M	x_6	20	1	2	1	0	0	1
Δ_j		$90M$	$4M+4$	$5M+1$	$9M+3$	0	0	0

Задачата е за търсене на минимум. Най-голямата Δ е $\Delta_3 = 9M + 3$. Преминаваме към друг опорен план.

C	$Б_x$	A_0	-4	-1	-3	M	M
			A_1	A_2	A_3	A_4	A_6
M	x_4	6	$-1/5$	**7/5**	0	1	0
-3	x_3	8	$2/5$	$1/5$	1	0	0
M	x_6	12	$3/5$	$9/5$	0	0	1
Δ_j		$18M-24$	$2M/5-14/5$	$16M/5+2/5$	0	0	0

Изкуствената променлива x_5 изпълни предназначението си (участва в създаването на изкуствения базис). Тя вече не е необходима при решаване на задачата и съответният ѝ вектор-стълб A_5 е изключен от таблицата.

C	$Б_x$	A_0	-4	-1	-3	M
			A_1	A_2	A_3	A_6
-1	x_2	$30/7$	$-1/7$	1	0	0
-3	x_3	$50/7$	$3/7$	0	1	0
M	x_6	$30/7$	**6/7**	0	0	1
Δ_j		$30M/7-180$	$6M/7+20/7$	0	0	0

Критерият за оптималност не е налице, $\Delta_1 = \frac{6}{7}M + \frac{20}{7} > 0$. Преминаваме към друг опорен план.

C	$Б_x$	A_0	-4	-1	-3
			A_1	A_2	A_3
-1	x_2	5	0	1	0
-3	x_3	5	0	0	1
-4	x_1	5	1	0	0
Δ_j		-40	0	0	0

Полученият план $X_{\text{опт}} = (5, 5, 5, 0, 0, 0)$ е оптимален за разширената задача и от теорема 1.5 следва

$$\min L(X) = L(X_{\text{опт}}) = -40.$$

ПРИМЕР 1.5. Да се намери моделът на разширената задача, ако изходната задача е:

$$L(X) = x_1 + 2x_2 - 3x_3 \to \begin{matrix}\max\\ \min\end{matrix}$$

при условия:

$$\begin{array}{rrrl} x_1 & +4x_2 & -x_3 & \le 8 \\ -x_1 & & +2x_3 & \ge 3 \\ 3x_1 & -5x_2 & & = 7 \end{array}$$

$$x_j \ge 0, \ j = 1, \ 2, \ 3.$$

Решение: Ограниченията са:

$$\begin{array}{rrrrrrrl} x_1 & +4x_2 & -x_3 & +x_4 & & & & = 8 \\ -x_1 & & +2x_3 & & -x_5 & +x_6 & & = 3 \\ 3x_1 & -5x_2 & & & & & +x_7 & = 7 \end{array}$$

а целевата функция е:

$$L(X) = x_1 + 2x_2 - 3x_3 + 0.x_4 - 0.x_5 \mp Mx_6 \mp Mx_7.$$

Променливите x_4 и x_5 са допълнителни, а променливите x_6 и x_7 са изкуствени.

Изходният базис на M-задачата е (x_4, x_6, x_7), като за базисна променлива е използвана допълнителната променлива x_4 и въведените изкуствени променливи x_6 и x_7.

Начален опорен план е $\overline{X} = (0, 0, 0, 8, 0, 3, 7)$ и $L(\overline{X}) = \mp 3M \mp 7M = \mp 10M.$

Съществува *модифициран симплекс-метод*, при който обемът на изчисленията е по-малък. При този метод се преобразува само обратната матрица на матрицата от базисни вектори и векторът-решение. Този метод ние няма да разглеждаме.

8. Дуална задача. Основна теорема на линейното оптимиране

Линейното оптимиране е от онези направления в математиката, за които е получен основен централен резултат, който в значителна степен определя съдържанието на цялата теория и влияе на други раздели от математиката. Така наречената *Теорема за дуалност (двойнственост)* безусловно е един такъв резултат. Неговото значение не се ограничава само в рамките на линейното оптимиране, а влияе както на формирането на методология в общата теория на оптимизационните задачи, така и на някои нематематически дисциплини.

Нека разгледаме един класически пример за линейна оптимизационна задача. Да се намери

$$L(X) = 2x_1 + 4x_2 + x_3 \to \max \tag{1.26}$$

при ограничения

$$\begin{aligned} 2x_1 + \ \ x_2 + 3x_3 &\le 100, \\ x_1 + 2x_2 + 2x_3 &\le 190, \\ 2x_1 + \ \ x_2 + 4x_3 &\le 120, \\ x_1 \ge 0,\ x_2 \ge 0,\ x_3 &\ge 0. \end{aligned} \tag{1.27}$$

Да разгледаме този модел като интерпретация на следната икономическа задача. В 3 цеха с общи капацитети (мощности), съответно 100, 190, 120 машиночаса, се произвеждат 3 вида артикули x_1, x_2 и x_3. Коефициентите на векторите-стълбове в матрицата на ограниченията задават изразходваните часове на различните цехове за производството на единица продукция от съответния артикул. Коефициентите на целевата функция посочват печалбата от единица продукция на всеки от артикулите. В тази задача се търси производствена програма (x_1, x_2, x_3), при която се получава максимална доходност (икономическа изгода).

Нека сега с y_1, y_2 и y_3 обозначим цените на 1 машиночас в трите цеха. Тогава

$$2y_1 + y_2 + 2y_3$$

може да се интерпретира като общи разходи на трите цеха за производството на единица продукция от първия вид. Аналогично

$$\begin{aligned} y_1 + 2y_2 + \ \ y_3&, \\ 3y_1 + 2y_2 + 4y_3& \end{aligned}$$

са съответно разходите за производство на единица продукция от втория и третия вид.

Да предположим, че цените y_1, y_2, y_3 на машиночасовете в трите цеха са избрани така, че

$$\begin{aligned} 2y_1 + \ \ y_2 + 2y_3 &\ge 2, \\ y_1 + 2y_2 + \ \ y_3 &\ge 4, \\ 3y_1 + 2y_2 + 4y_3 &\ge 1, \end{aligned}$$

(сумарните разходи за производството на единица от всеки артикул трябва да надхвърля печалбата от него).

И тъй като 100, 190, 120 са ресурсите машинно време на всеки от цеховете, то

$$L(Y) = 100y_1 + 190y_2 + 120y_3$$

ще изразява сумарните разходи за производство. Задачата

$$L(Y) = 100y_1 + 190y_2 + 120y_3 \to \min \tag{1.28}$$

при ограничения

$$\begin{aligned} 2y_1 + \ y_2 + 2y_3 &\geq 2, \\ y_1 + 2y_2 + \ y_3 &\geq 4, \\ 3y_1 + 2y_2 + 4y_3 &\geq 1, \\ y_1 \geq 0,\ y_2 \geq 0,\ y_3 &\geq 0, \end{aligned} \tag{1.29}$$

се нарича *дуална (двойнствена, спрегната)* на задачата (1.26) — (1.27).

Най-общо всяка задача на линейното оптимиране е свързана с друга задача, наречена дуална. Съществува връзка между решенията на двете задачи. Първата задача ще наричаме понякога *начална (изходна, пряка)*. Променливите на дуалната задача се наричат още *скрити цени*. Ще дадем дефиниция за дуална задача на общата задача на линейното оптимиране.

ОПРЕДЕЛЕНИЕ 1.3. *Нека* $M = \{1, 2, ..., m\}$, $N = \{1, 2, ..., n\}$ *и* $L \subset M$, $K \subset N$. *Пряка задача:* (1.30) $L(X) = c_1x_1 + c_2x_2 + ... + c_nx_n \to \max$ $a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \leq b_i,\ i \in L$ $a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i,\ i \in M \setminus L$ $x_i \geq 0,\ i \in K$ *Дуална задача:* (1.31) $L(Y) = b_1y_1 + b_2y_2 + ... + b_my_m \to \min$ $a_{1j}y_1 + a_{2j}y_2 + ... + a_{mj}y_m \geq c_j,\ j \in K$ $a_{1j}y_1 + a_{2j}y_2 + ... + a_{mj}y_m = c_j,\ j \in N \setminus K$ $y_j \geq 0,\ j \in L$	**Обща форма на дуалната задача**

ЗАДАЧА 1.10. Да се докаже, че

	Дуалната задача на дуалната съвпада с пряката (в една двойка дуални задачи всяка може да бъде третирана като пряка).

От казаното дотук следва, че основните правила за получаване на дуалната задача от пряката са:

1. На всяко ограничение, представено чрез неравенство, отговаря дуална променлива, подчинена на условия за неотрицателност, а на всяка неотрицателна променлива отговаря ограничително неравенство в дуалната задача.

2. На всяко ограничение от тип равенство отговаря свободна дуална променлива (произволна по знак), а на всяка свободна променлива отговаря ограничение от тип равенство в дуалната задача.

3. Матриците от условията на две дуални задачи са транспонирани една на друга.

4. Свободните членове на ограниченията на всяка от две дуално спрегнати задачи съвпадат с коефициентите на целевата функция в другата задача.

5. Ако в едната задача се търси максимум, в нейната дуална се търси минимум и обратно.

Очевидно, при $L = M$ и $K = N$ (в пряката задача всички ограничения са от типа "$\leq$" и всички променливи x_i са неотрицателни) двойката дуални задачи изглежда така:

Пряка задача:

$$L(X) = c_1x_1 + c_2x_2 + ... + c_nx_n \to \max$$

$$(1.32)\qquad \begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &\leq b_1,\\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &\leq b_2,\\ \dots\dots\dots\dots\dots\dots\dots\dots\dots&\dots\dots\\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &\leq b_m,\\ x_j \geq 0,\ j = 1,\ 2, ...,\ n.& \end{aligned}$$

Дуална задача:

$$D(Y) = b_1y_1 + b_2y_2 + ... + b_my_m \to \min,$$

$$(1.33)\qquad \begin{aligned} a_{11}y_1 + a_{21}y_2 + ... + a_{m1}y_m &\geq c_1,\\ a_{12}y_1 + a_{22}y_2 + ... + a_{m2}y_m &\geq c_2,\\ \dots\dots\dots\dots\dots\dots\dots\dots\dots&\dots\dots\\ a_{1n}y_1 + a_{2n}y_2 + ... + a_{mn}y_m &\geq c_n,\\ y_i \geq 0,\ i = 1,\ 2, ...,\ m.& \end{aligned}$$

Симетрични дуални задачи

Очевидно, при $L = \emptyset$ и $K = N$ (пряката задача е в каноничен вид), двойката дуални задачи изглежда така:

Пряка задача: $$L(X) = c_1x_1 + c_2x_2 + \dots + c_nx_n \to \max$$ (1.34) $$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m, \\ x_j \geq 0,\ j = 1,\ 2, \dots,\ n. \end{aligned}$$ *Дуална задача:* $$D(Y) = b_1y_1 + b_2y_2 + \dots + b_my_m \to \min,$$ (1.35) $$\begin{aligned} a_{11}y_1 + a_{21}y_2 + \dots + a_{m1}y_m \geq c_1, \\ a_{12}y_1 + a_{22}y_2 + \dots + a_{m2}y_m \geq c_2, \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ a_{1n}y_1 + a_{2n}y_2 + \dots + a_{mn}y_m \geq c_n, \\ y_i - \text{произволни по знак},\ i = 1, 2, \dots, m. \end{aligned}$$	**Несиметрични дуални задачи**

Векторният запис на симетричните и несиметричните дуални задачи е:

Симетрични дуални задачи		**Несиметрични дуални задачи**	
Пряка:	Дуална:	Пряка:	Дуална:
$L(X)=(C,X)\to\max$	$D(Y)=(Y,B)\to\min$	$L(X)=(C,X)\to\max$	$D(Y)=(Y,B)\to\min$
$AX \leq B$	$YA \geq C$	$AX = B$	$YA \geq C$
$X \geq 0$	$Y \geq 0$	$X \geq 0$	$Y \in \mathbb{R}$

ПРИМЕР 1.6.

Пряка задача

$$L(X) = x_1 + 2x_2 \to \max$$

при ограничения

$$\begin{aligned} x_1 \quad + x_2 &\leq 3 \\ 2x_1 \quad - x_2 &\leq 4 \\ 3x_1 \quad + 4x_2 &\leq 17 \\ -x_1 \quad - 6x_2 &\geq 1 \\ 5x_1 \quad - 3x_2 &\geq 5 \\ x_1 \geq 0,\ x_2 &\geq 0. \end{aligned}$$

Дуална задача

$$D(Y) = 3y_1 + 4y_2 + 17y_3 - y_4 - 5y_5 \to \min$$

при ограничения

$$y_1 + 2y_2 + 3y_3 + y_4 - 5y_5 \geq 1$$

$$y_1 - y_2 + 4y_3 + 6y_4 + 3y_5 \geq 2$$

$$y_i \geq 0, \; i = 1, \; 2, \; 3, \; 4, \; 5.$$

9. Теореми за двойнственост

ЛЕМА 1.1 Ако $\overline{X}$ и $\overline{Y}$ са произволни планове, съответно на задачите (1.34) и (1.35), то

$$L(\overline{X}) \leq D(\overline{Y})$$

Верността на лемата следва от $L(\overline{X}) = C\overline{X} \leq \overline{Y}A\overline{X} = \overline{Y}B = D(\overline{Y})$. Очевидно, ако планът $\overline{Y}$ не е оптимален, то $L(\overline{X}) < D(\overline{Y})$.

ЛЕМА 1.2. (Критерий за оптималност) *Ако X' и Y' са произволни планове, съответно на задачи (1.34) и (1.35), за които*

$$L(X') = D(Y'),$$

то X' и Y' са оптимални.

ЛЕМА 1.3. Ако целевата функция $D(Y)$ на дуалната задача (1.35) е неограничена отдолу, то множеството от планове на пряката задача (1.34) е празно.

Последните 2 твърдения са очевидни следствия от лема 1.1.

Помощните твърдения, които доказахме, очевидно са верни и за двойка симетрични дуални задачи.

▷ *ТЕОРЕМА 1.6.* (Фундаментална теорема) *Двойката дуални задачи (1.34) и (1.35) са едновременно разрешими или неразрешими.*

Ако $\overline{X}$ и $\overline{Y}$ са съответно оптимални планове на тези задачи, то

$$L_{\max} = L(\overline{X}) = D(\overline{Y}) = D_{\min}.$$

◁

Теорема 1.6 е в сила за всяка двойка дуални задачи.

От лема 1.2 и теорема 1.6 следва:

Необходимо и достатъчно условие за оптималност на плановете $\overline{X}$ и $\overline{Y}$ е

$$L(\overline{X}) = D(\overline{Y}).$$

Ако $\overline{X}$ е оптимален план на пряката задача (1.34) и B_0^{-1} е обратната матрица на базиса на този план, а $\tilde{C}$ е векторът от коефициентите пред базисните променливи в $L(X)$, то $\overline{Y} = \tilde{C}B_0^{-1}$ е оптимален план на дуалната задача.

Ако пряката задача се решава чрез обикновен симплекс-метод, компонентите на оптималния план $\overline{Y}$ се получават от "оптималния" индексен ред — към Δ-те съответни на векторите на първоначалния базис се прибавят съответните числа — коефициенти от целевата функция.

ПРИМЕР 1.7. Да се реши дуалната на следната задача

$$L(X) = 4x_1 + 3x_2 + 5x_3 - 20x_4 \to \max$$

при условия

$$\begin{array}{rrrrl} x_1 & +8x_2 & +7x_3 & -15x_4 & = 17 \\ -x_1 & +5x_2 & +6x_3 & -11x_4 & = 9 \end{array}$$

$$x_j \geq 0, \; j = 1, \; 2, \; 3, \; 4.$$

Дуалната задача на тази задача е

$$D(Y) = 17y_1 + 9y_2 \to \min$$

при условия

$$\begin{array}{rrl} y_1 & -\ y_2 & \geq 4 \\ 8y_1 & +\ 5y_2 & \geq 3 \\ 7y_1 & +\ 6y_2 & \geq 5 \\ -15y_1 & -11y_2 & \geq -20 \end{array}$$

y_i – произволни по знак.

В този, както и в други случаи, при които броят на ограниченията в една от двете дуални задачи е по-голям от броя на неизвестните, по-удачно е да се решава онази, която има по-малък брой ограничения. В случая по-удачно е да се реши пряката задача и от нея да се вземе решението на дуалната.

Преобразуваме пряката задача във вид, удобен за прилагане на симплекс-метода

$$L(X) = 4x_1 + 3x_2 + 5x_3 - 20x_4 - Mx_5 - Mx_6$$

при ограничения

$$\begin{array}{l} x_1 + 8x_2 + 7x_3 - 15x_4 + x_5 = 17 \\ -x_1 + 5x_2 + 6x_3 - 11x_4 + x_6 = 9 \end{array}$$

$$x_j \geq 0,\ j = 1,\ 2, \ldots,\ 6.$$

C	$Б_x$	A_0	4	3	5	−20	−M	−M
			A_1	A_2	A_3	A_4	A_5	A_6
$-M$	x_5	17	1	8	7	−15	1	0
$-M$	x_6	9	−1	5	**[6]**	−11	0	1
Δ_j		$-26M$	−4	$-13M-3$	$-13M-5$	$26M+20$	0	0

C	$Б_x$	A_0	4	3	5	−20	−M	−M
			A_1	A_2	A_3	A_4	A_5	A_6
$-M$	x_5	13/2	**[13/6]**	13/6	0	−13/6	1	−7/6
5	x_3	3/2	−1/6	5/6	1	−11/6	0	1/6
Δ_j		$-13M+15/2$	$-13M-29/6$	$-13M+7/6$	0	$13M+65/6$	0	$13M+5/6$

C	$Б_x$	A_0	4	3	5	−20	−M	−M
			A_1	A_2	A_3	A_4	A_5	A_6
4	x_1	3	1	1	0	−1	6/13	−7/13
5	x_3	2	0	1	1	−2	1/13	1/13
Δ_j		22	0	6	0	6	$13M+29/13$	$13M-23/13$

Полученият план $\overline{X} = (3, 0, 2, 0)$ е оптимален за пряката задача и $L_{\max} = 22$.
От направените теоретични бележки следва:

1. $D_{\min} = L_{\max} = 22$.
2. Оптималният план е $\overline{Y} = (\bar{y}_1, \bar{y}_2)$, където

$$\bar{y}_1 = \frac{13M + 29}{13} - M = \frac{29}{13},$$

$$\bar{y}_2 = \frac{13M - 23}{13} - M = -\frac{23}{13},$$

$$\text{т.е. } \overline{Y} = \left(\frac{29}{13}, -\frac{23}{13}\right).$$

ЛЕМА 1.4. Необходимо и достатъчно условие плановете $\overline{X}$ и $\overline{Y}$ съответно на дуалните задачи (1.32) и (1.33) да са оптимални е

$$Y(B - AX) = 0 \quad \text{и} \quad (YA - C)X = 0. \tag{1.36}$$

Да разпишем равенствата (1.36) по-подробно.

$$\overline{Y}(B - A\overline{X}) = \sum_{i=1}^{m} \bar{y}_i(b_i - a_{i1}\bar{x}_1 - a_{i2}\bar{x}_2 - \ldots - a_{in}\bar{x}_n) = 0,$$

$$(\overline{Y}A - C)\overline{X} = \sum_{j=1}^{n} (a_{1j}\bar{y}_1 + a_{2j}\bar{y}_2 + \ldots + a_{mj}\bar{y}_m - c_j)\bar{x}_j = 0.$$

Тъй като всяко събираемо в горните две суми е неотрицателно (произведение от два неотрицателни множителя), а сумите са равни на нула, следва

$$\forall i, \ \bar{y}_i(b_i - a_{i1}\bar{x}_1 - a_{i2}\bar{x}_2 - \ldots - a_{in}\bar{x}_n) = 0,$$

$$\forall j, \ (a_{1j}\bar{y}_1 + a_{2j}\bar{y}_2 + \ldots + a_{mj}\bar{y}_m - c_j)\bar{x}_j = 0.$$

Оттук веднага следват релациите

1. Ако $\bar{y}_i > 0, \ i \in \{1, 2, \ldots, m\} \Longrightarrow a_{i1}\bar{x}_1 + a_{i2}\bar{x}_2 + \ldots + a_{in}\bar{x}_n = b_i$;
2. Ако $a_{i1}\bar{x}_1 + a_{i2}\bar{x}_2 + \ldots + a_{in}\bar{x}_n < b_i, \ i \in \{1, \ 2, \ldots, \ m\} \Longrightarrow \bar{y}_i = 0$;

3. Ако $\bar{x}_j > 0,\ j \in \{1,\ 2, \ldots,\ n\} \Longrightarrow a_{1j}\bar{y}_1 + a_{2j}\bar{y}_2 + \ldots + a_{mj}\bar{y}_m = c_j$;

4. Ако $a_{1j}\bar{y}_1 + a_{2j}\bar{y}_2 + \ldots + a_{mj}\bar{y}_m > c_j,\ j \in \{1,\ 2, \ldots,\ n\} \Longrightarrow \bar{x}_j = 0$.

Направените по-горе бележки можем да обобщим в следното твърдение.

▷ *ТЕОРЕМА 1.7.* (Теорема за равновесието) *Ако i-тата компонента на оптималния план на една от двойката дуални симетрични задачи е положителна, то i-тото ограничение в дуалната задача за оптималния си план е изпълнено като равенство.*
Ако i-тото ограничение в едната задача е изпълнено за оптималния си план, като строго неравенство, то i-тата компонента на оптималния план на дуалната задача е нула. ◁

Втората теорема за дуалност, която доказахме, се тълкува аналогично за общата форма на дуалните задачи (1.30) и (1.31).

При несиметрични дуални задачи (1.34) — (1.35) теоремата за равновесието включва релациите 3 и 4, формулирани преди теорема 1.7.

Икономическата интерпретация на изходната (пряката) задача е ясна. Пряката задача се състои в това да се определи оптимален производствен план, така че от реализацията на произведената продукция да се получи максимален доход (печалба).

Икономическият смисъл на двойствената задача е да се определят обективни оценки на единица от всеки ресурс i, които при дадените ресурси b_i и дадените доходи c_j, получавани от реализацията на единица продукция от вид j, минимизират общата стойност на разходите.

В заключение ще отбележим, че икономическата интерпретация на получените два основни резултата в този параграф — теорема 1.6 и теорема 1.7, е следната.

Оптимален производствен план съществува само когато всички производствени ресурси b_i имат оценка (т.е. когато дуалната задача е разрешима) и обратно.

Освен това (ако дуалната задача е разрешима), оценката на реализирания продукт съвпада с общата оценка на наличните ресурси.

Смисълът на теорема 1.7 е, че ако за някой ресурс b_i при оптималния план ограничението за този ресурс се изпълнява като строго неравенство, то той се явява несъществен (недефицитен) и цената, съответстваща на ресурса, е равна на нула.

Някои задачи от теория на графите могат успешно да се решават със симплекс-метода или с прилагане на аналогични техники. А именно, стартира се от някакво базисно решение и от него се преминава към друго базисно решение, подобряващо значението на целевата функция. След краен брой итерации на процедурата се стига до оптималното решение. В това ще се убедим с разглежданията в следващите параграфи на тази глава.

В много случаи обаче симплекс-методът е неприложим или прилагането му е твърде неефективно и води до "тромави" решения. Типичен пример в това отношение е т. нар.

Класическа транспортна задача: Продукцията $a_1, a_2, \dots, a_m$ на производителите $A_1, A_2, \dots, A_m$, трябва да бъде разпределена между пунктовете $B_1, B_2, \dots, B_n$ с потребление $b_1, b_2, \dots, b_n$, като

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

и транспортните разходи за превоза на единица продукт от A_i до B_j са c_{ij}. Да се направи такъв план за разпределяне на продукцията в пунктовете $B_1, \dots, B_n$, така че техните потребности да бъдат изцяло задоволени и общите транспортни разходи по превоза на цялата продукция да бъдат минимални.

Математически модел на задачата:

$$L(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \to \min$$

$$\sum_{j=1}^{n} x_{ij} = a_i, \qquad i = 1, 2, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} = b_j, \qquad j = 1, 2, \dots, n$$

$$x_{ij} \geq 0, \qquad i = 1, \dots, m, \; j = 1, \dots, n$$

Очевидно това е задача на линейното оптимиране и може да се решава със симплекс-метода. Това обаче ще наподобява

"стрелба с оръдие по врабче". За целта е разработен специален метод (отчитащ спецификата на конкретната задача), наречен *разпределителен метод*, който е значително по-ефективен.

Освен това, при нелинейни задачи симплекс-методът не е приложим.

За много задачи в теория на графите са разработени специални методи, водещи до решения на задачи с големи размери в реално време (нещо, което симплекс-методът не реализира). Такива задачи и алгоритми ще разгледаме в следващите параграфи на тази глава. Накрая ще отбележим (припомним) следното.

Понякога на всяка дъга (ребро) на графа G се приписва (съпоставя) число, наречено *"тегло"*, *"дължина"*, *"стойност"*, *"цена"* на дъгата. Приписаните на всяка дъга (ребро) числа могат да са няколко и освен на дъгите (ребрата), съответни числа могат да бъдат съпоставени и на върховете. В различните случаи, в зависимост от естеството на задачата се употребява най-близката по смисъл дума. Например при търсене на минимални (най-кратки) пътища, теглото на всяка дъга наричаме *"дължина на дъгата"*.

Граф, чийто дъги (ребра) или върхове имат съответни тегла, се нарича *мрежа*.

Най-често теглото на дъгата (v_i, v_j) ще бележим със $c(v_i, v_j)$, c_{ij} или $c(e_k)$, където $e_k = (v_i, v_j)$, както и с $d(v_i, v_j)$, $p(v_i, v_j)$.

Ако ρ е някакъв път, например път от дъгите $e_1, e_2, \dots, e_k$, под *тегло (дължина, стойност, цена) на пътя* ρ се разбира най-често сумата от теглата (дължините, стойностите, цените) $c(e_i)$ на неговите дъги, т.е.

$$c(\rho) = \sum_{e_i \in \rho} c(e_i).$$

Ще припомним, че когато става въпрос за броя на участващите в даден път (или верига) дъги, ще употребяваме термина мощност на пътя. Ясно е в кои случаи дължината на пътя съвпада с мощността.

2.2. Алгоритми за построяване на покриващи дървета

Много реални задачи в областта на икономиката, управлението, обслужващата сфера и др. могат да се интерпретират

като задачи за намиране (търсене, построяване) на покриващи дървета, удовлетворяващи някакви условия за оптималност.

ПРИМЕР 2.1. (Задача за клюката). В един университет някои от преподавателите ежедневно се срещат и разговарят, като споделят помежду си всеки интересен слух. Може ли в този университет, т.е. между всички преподаватели, да бъде разпространена някаква клюка.

Поставеният проблем само на пръв поглед звучи несериозно, защото най-общо това е проблем, свързан с управлението и разпространението на информация, рекламна дейност и др. Не е трудно да се съобрази, че може да става въпрос за разпространение на стоки до всички търговски обекти, на суровини до производствени предприятия, в зависимост от пътната мрежа и т.н.

Нека всеки преподавател разглеждаме като връх на един неориентиран граф G, на който ребрата показват кои от преподавателите ежедневно се срещат и разговарят. По своята същност проблемът е дали този граф е свързан? Отговорът на този въпрос е положителен, ако може да се построи покриващо дърво за този граф. Невъзможността да се построи покриващо дърво за графа означава и невъзможност за разпространение на слуха в целия университет.

В други задачи се налага не да се намери някакво покриващо дърво изобщо, а да се търси дърво с оптимални свойства. Нека в неориентирания граф $G = (V, A)$ всяко ребро (u, v) има тегло $c(u, v)$, като под *тегло на дървото* се разбира сумата от теглата на участващите в него ребра.

ПРИМЕР 2.2. Строителна фирма разглежда проект за строеж на пътища между шест селища, който трябва да осигури комуникациите между тези населени места. Известни са разходите (цените), необходими за прокарването на всеки от възможните пътища между тези селища. Фирмата иска с минимални разходи да осигури пътните комуникации (не е задължително да се прокарва път между всеки две селища).

Да разгледаме неориентирания граф $G = (V, A)$, чийто върхове съответстват на градовете, ребрата — на пътищата, които могат да се прокарат между тези градове, а теглата на ребрата са съответните разходи за прокарване на пътя. Ясно е, че в случая стратегията на фирмата се свежда до намиране на покриващо дърво за този граф с минимално тегло (разходи, цени). Да припомним, че покриващото дърво включва всички върхове на графа (вж. предишния параграф), което осигурява път между всеки два града.

Разбира се в реални ситуации нещата са доста по-сложни, тъй като освен критерия минималност на инвестициите, се държи сметка за скоростта на придвижване, пропускателната способност, надеждността на пътната мрежа и много други фактори, които в случая са игнорирани.

1. Алгоритъм за построяване на покриващо дърво

Този алгоритъм е едно от многото красиви, изящни и елегантни неща в математиката, които изненадват със своята простота, яснота — каквито са всъщност повечето от дълбоките и сериозни идеи в математиката.

ИДЕЯ НА АЛГОРИТЪМА. В произволен ред се разглеждат ребрата на изходния граф, като на всяка стъпка на алгоритъма се взима решение съответното ребро да бъде включено или не в покриващото дърво. При това реброто, което се включва в дървото, се оцветява в *зелено*, а това, което не се включва — в *черно*, т.е. алгоритъмът е процес на *оцветяване на ребрата.* При това, в този алгоритъм се взема еднозначно и окончателно решение — реброто се оцветява в един от избраните цветове и по-нататък не е обект на разглеждане. Такива алгоритми се наричат *поглъщащи* и имат две важни характеристики — не се прави разход на време за повторни (последващи) разглеждания и лесно се определя максималният брой изпълними операции (стъпки) на алгоритъма.

На всяка стъпка в алгоритъма се прави проверка, дали разглежданото ребро в съвкупност със зелените (т.е. включените вече в дървото) ребра образува цикъл. Ако това е така — реброто се оцветява в черно (т.е. не се включва в покриващото дърво), в противен случай се оцветява в зелено (т.е. се включва в покриващото дърво).

Какво обаче означава, т.е. как се извършва "проверка дали разглежданото ребро образува цикъл с включените вече в дървото ребра". Зелените (включените в дървото) ребра образуват един или повече свързани компоненти. Върховете на всеки от компонентите образуват множество от върхове, което ще наричаме *"букет"*. Следователно разглежданото ребро ще образува цикъл с включените в дървото ребра, ако и двата му върха принадлежат на един от формираните до момента букети. С други думи, освен оцветяването на ребрата, в алгоритъма се поддържат (съхраняват) и актуализират букети от върхове на отделни свързани компоненти, което прави възможна проверка-

та за съществуване на цикъл.

Алгоритъмът приключва своята работа, когато всички n върха на графа попаднат в един букет, т.е. ребрата, включени в строеното дърво, се окажат еднокомпонентен (свързан) граф без цикли, включващ всички върхове на изходния граф. Еквивалентно на това условие за край на алгоритъма е условието броят на зелените ребра да бъде $n-1$. Това наистина е така, защото (вж. предишната глава) всяко покриващо дърво на граф с n върха има $n-1$ ребра.

ОПИСАНИЕ НА АЛГОРИТЪМА ЗА НАМИРАНЕ НА ПОКРИВАЩО ДЪРВО. *СТЪПКА 1.* Избираме произволно ребро (което не е примка). Оцветяваме това ребро в зелено. Сформираме букет от върховете на това ребро.

СТЪПКА 2. Избираме произволно неоцветено ребро (което не е примка). Ако такова ребро няма, преминаваме към *стъпка 4.*

СТЪПКА 3. Възможни са следните четири случая:

а) нито един от краищата на реброто не принадлежи на сформиран до момента букет — оцветяваме реброто в зелено и сформираме нов букет от върховете (краищата) на това ребро. Преминаваме към *стъпка 4*;

б) и двата края на реброто принадлежат на един и същ букет върхове, сформиран до момента — оцветяваме реброто в черно, т.е. не го включваме в дървото, което строим. Преминаваме към *стъпка 4*;

в) единият връх на реброто принадлежи на даден букет, а другият не принадлежи на никой от сформираните букети — оцветяваме реброто в зелено и невключения в букет връх, включваме в букета, съдържащ другия връх. Преминаваме към *стъпка 4*;

г) върховете на реброто принадлежат на различни букети — оцветяваме реброто в зелено и обединяваме двата букета в един нов букет. Преминаваме към *стъпка 4.*

СТЪПКА 4. Ако всички върхове на графа са в един букет (броят на оцветените ребра е с единица по-малко от броя на върховете на графа), ребрата, оцветени в зелено, образуват покриващо дърво за графа. Край.

СТЪПКА 5. Преминаваме към *стъпка 2.*

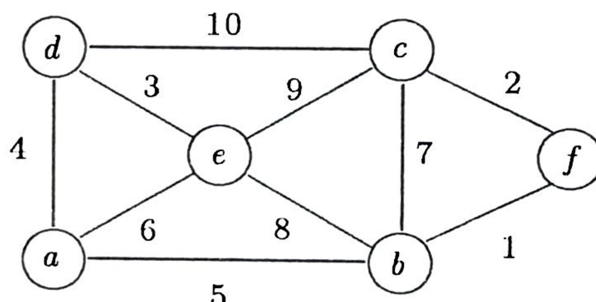
Очевидно, така формулираният алгоритъм "работи" добре, когато съществува покриващо дърво и "зациклят", ако такова не съществува.

ЗАДАЧА 2.1. Да се премахне "зациклянето" в изложения алгоритъм за намиране на покриващо дърво (когато такова няма), така че алгоритъмът да установява несъществуването на покриващо дърво и да спира своята работа.

ЗАДАЧА 2.2. Да се даде горна оценка за броя на стъпките (операциите) при изпълнението на алгоритъма за търсене на покриващо дърво, в зависимост от броя на ребрата на графа.

Ще илюстрираме алгоритъма със следния пример.

ПРИМЕР 2.3. Даден е граф $G = (V, A)$



Да се построи покриващо дърво за този граф. Резултатите от прилагането на алгоритъма ще изложим за прегледност в таблица (което не е задължително).

No	Ребро	Цвят	Букет 1 (от върхове)	Букет 2 (от върхове)	Букет 3 (от върхове)	...
1	(a, e)	зелен	a, e	$\emptyset$	$\emptyset$	
2	(b, f)	зелен	a, e	b, f	$\emptyset$	
3	(d, c)	зелен	a, e	b, f	d, c	
4	(e, c)	зелен	обединяваме букети 1 и 3 a, e, d, c	b, f	$\emptyset$	
5	(a, d)	черен	a, e, d, c	b, f	$\emptyset$	
6	(e, b)	зелен	a, e, d, c, b, f	$\emptyset$	$\emptyset$	

Тъй като след разглеждане и оцветяване на шестото ребро всички върхове на графа се оказват в един букет (или което е същото — броят на оцветените в зелено ребра се оказва с единица по-малък от броя на върховете на графа), дървото, състоящо се от ребрата (a, e), (b, f), (d, c), (e, c), (e, b), е покриващо дърво за изходния граф.

Лесно се проверява, че ако разглеждаме ребрата на графа в друг ред, ще получим други покриващи дървета (проверете).

Нека числата, приписани на всяко ребро от графа в пример 2.3, са съответните тегла на ребрата. Намереното покриващо дърво в таблицата може да се разглежда като една от възможните стратегии на строителната фирма, цитирана в пример 2.2. Ако теглата на ребрата са разходите в млн. лева, за построяването на съответните участъци от пътната мрежа, лесно се установява, че това не е най-доброто решение за фирмата по отношение общия обем на инвестициите.

Теглото на намереното покриващо дърво е (общите разходи в млн. лв.)

$$c(a,e)+c(b,f)+c(d,c)+c(e,c)+c(e,b) = 6+1+10+9+8 = 34 \text{ млн. лв.}$$

Интуицията подсказва, че ако ребрата бяха разглеждани не в произволен ред, а по реда на нарастване на техните тегла (цени), така полученото покриващо дърво ще има по-малко тегло от предишното. Покриващото дърво, състоящо се от ребрата

$$(f,b),\ (b,c),\ (b,a),\ (a,d),\ (d,e),$$

има тегло $1+7+5+4+3 = 20$, но и то не е *минимално покриващо дърво* — покриващо дърво с минимално тегло, т.е. покриващо дърво, чието тегло не е по-голямо от теглото на кое да е друго покриващо дърво. Да разгледаме ребрата на графа от пример 2.3 по реда на нарастване на техните тегла и приложим алгоритъма за търсене на покриващо дърво.

No	Ребро	Цвят	Букет 1 (от върхове)	Букет 2 (от върхове)	...
1	(f,b)	зелен	f,b	$\emptyset$	
2	(f,c)	зелен	f,b,c	$\emptyset$	
3	(d,e)	зелен	f,b,c	d,e	
4	(d,a)	зелен	f,b,c	d,e,a	
5	(a,b)	зелен	f,b,c,d,e,a	$\emptyset$	

Полученото покриващо дърво, състоящо се от ребрата във втория стълб на таблицата, има тегло $1+2+3+4+5 = 15$ и очевидно то е минимално покриващо дърво.

Ако теглата на ребрата в графа от пример 2.3 изразяваха не разходи, а печалба в млн. лв., тогава естествено е да се търси

максимално покриващо дърво — покриващо дърво, чието тегло е не по-малко от теглото на което и да е покриващо дърво. За целта се оказва достатъчно да приложим алгоритъма за намиране на покриващо дърво, като ребрата на графа разглеждаме по реда на намаляване на техните тегла (започваме от реброто с най-голямо тегло). Можем да формулираме следните две твърдения.

1. *Алгоритъм за търсене на минимално покриващо дърво.*
Прилага се алгоритъмът за търсене на покриващо дърво, като ребрата се разглеждат по реда на нарастване на теглата им. Ако има ребра с еднакви тегла, те се разглеждат в произволен ред.

2. *Алгоритъм за търсене на максимално покриващо дърво.*
Прилага се алгоритъмът за търсене на покриващо дърво, като ребрата се разглеждат по реда на намаляване на техните тегла. Ако има ребра с еднакви тегла, те се разглеждат в произволен ред.

ЗАДАЧА 2.3. Да се намери максималното покриващо дърво за графа от пример 2.3.

ЗАДАЧА 2.4. Да се покаже, че алгоритъмът за търсене на минимално покриващо дърво може да се използва за търсене на максимално покриващо дърво и обратно.
Упътване: Разгледайте граф с противоположни или реципрочни тегла на ребрата (игнорирайки ребрата с тегло нула).

ОБОСНОВКА НА АЛГОРИТЪМА *за намиране на максимално покриващо дърво.* (Алгоритъмът за търсене на минимално покриващо дърво се обосновава напълно аналогично или като се използва задача 2.4.)

Да означим с $T_{алг.}$ дървото, което алгоритъмът построява, а с T_{max} — максималното покриващо дърво. Да допуснем, че тези две дървета се различават поне по едно ребро и нека $e_1 = (u, v)$ е първото от разглежданите в алгоритъма ребра, което е от $T_{алг.}$, но не принадлежи на T_{max}.

Тъй като T_{max} е покриващо дърво, ще съществува единствен прост път $\rho(u, v)$, свързващ върховете u и v. Да добавим реброто e_1 към T_{max}. Тогава очевидно в T_{max} ще се появи цикъл, при това поне едно ребро на този цикъл няма да е от $T_{алг.}$ (в противен случай ще излезе, че в $T_{алг.}$ съществува цикъл, което е невъзможно). Да означим това ребро с e_2 и да изключим това ребро от T_{max}.

И така, в T_{max} включваме реброто e_1 и изключваме реброто e_2. Да означим с T^1_{max} полученото по този начин покриващо дърво. Тъй като T_{max} е максимално дърво, то

$$T^1_{max} \leq T_{max} \quad \text{(по определение)},$$

откъдето пък следва

(2.1) $$\text{теглото } (e_1) \leq \text{теглото } (e_2).$$

От друга страна, невключеното в $T_{алг.}$ ребро e_2 не може да е разглеждано в алгоритъма преди реброто e_1. Да допуснем противното, т.е. реброто e_2 е разглеждано в алгоритъма преди e_1. Тъй като e_2 не е включено в $T_{алг.}$, следва, че реброто e_2 е образувало цикъл с ребрата преди него, включени в $T_{алг.}$. Но тези ребра са и ребра на T_{max} (такива са всички ребра, разглеждани до e_1), следователно в T_{max} ще съществува цикъл, което е невъзможно. Отхвърляме допускането, т.е. e_2 не е разглеждано в алгоритъма преди реброто e_1. Тогава

(2.2) $$\text{теглото } (e_1) \geq \text{теглото } (e_2).$$

От (2.1) и (2.2) следва

$$\text{теглото } (e_1) = \text{теглото } (e_2),$$

което означава, че

(2.3) $$\text{теглото } (T^1_{max}) = \text{теглото } (T_{max}).$$

Освен това, общите ребра на T^1_{max} и $T_{алг.}$ са с единица повече от общите ребра между T_{max} и $T_{алг.}$.

Ако в качеството на максимално покриващо дърво сега разгледаме T^1_{max} (възможно е поради (2.3)) и повторим горните разсъждения, очевидно ще получим дърво T^2_{max}, общите ребра на което с $T_{алг.}$ ще са с единица повече, отколкото тези на T^1_{max} и

$T_{алг.}$. Така след краен брой стъпки, ще стигнем до максимално покриващо дърво T^k_{max}, което напълно съвпада с $T_{алг.}$.

Както споменахме в параграф 1.8, глава I, търсенето в дълбочина и в ширина може да се използва за намиране на покриващи дървета. Ще дадем едно по-формално описание на алгоритъм за построяване на покриващо дърво в свързан граф [44].

Данни: Свързан граф $G = (V, A)$, представен със списъци на инцидентност *ЗАПИС* $[v]$, $v \in V$.

Резултат: Покриващо дърво (V, T) в графа G.

```
1  procedure WGD(v)
   (*търсене в дълбочина с намиране ребрата на дървото;
   променливите НЕМАРКИРАН, ЗАПИС, T са глобални *)
2  begin НЕМАРКИРАН [v] := лъжа;
3     for u ∈ ЗАПИС [v] do
4        if НЕМАРКИРАН [u] then
         (* {v, u}-нов клон*)
5           begin T := T ∪ {v, u}; WGD(u)
6           end
7  end; (*WGD*)
8  begin (*главна програма*)
9     for u ∈ V do НЕМАРКИРАН [u] :=истина;
      (*инициализация*)
10       T := ∅
         (* T = множество от клоните, намерени до момента*)
11       WGD(r) (* r-произволен връх на графа*)
12 end
```

Предложената процедура WGD наистина построява дърво, защото:

1. Всеки път, когато в ред 5 на процедурата към множеството T се добавя нов клон $\{v, u\}$ във (V, T), съществува път от r до v (докажете с индукция). Следователно алгоритъмът строи свързан граф.

2. Всеки нов клон $\{v, u\}$, добавян към множеството T, съединява разгледан връх v (*НЕМАРКИРАН* $[v]$ =**лъжа**) с връх u, който е немаркиран. Следователно строеният граф (V, T) е ацикличен.

3. От свойствата на търсенето в дълбочина (вж. параграф 1.8) процедурата WGD обхожда всички върхове на свързания граф.

От 1, 2 и 3 следва, че графът (V,T), построяван с този алгоритъм, е свързан, ацикличен и включва всички върхове на графа, т.е. графът (V,T) е покриващо дърво за G.

Изчислителната сложност на този алгоритъм очевидно е $O(n+m)$, какъвто е порядъкът на търсенето в дълбочина.

Ще дадем и алгоритъм за намиране на покриващо дърво, използващ метода търсене в ширина [44].

```
begin
    for u ∈ V do НЕМАРКИРАН [u] := истина;
    (*инициализация*)
    T := ∅;
    (* T = множество от намерените до момента клони*)
    ОПАШКА = ∅; ОПАШКА ⟸ r;
    НЕМАРКИРАН [r] :=лъжа;
    (* r-корен на покриващо дърво*)
    while ОПАШКА ≠ ∅ do
        begin v ⟸ ОПАШКА;
            for u ∈ ЗАПИС [v] do
                if НЕМАРКИРАН [u] then (* {v, u} = нов клон*)
                    begin ОПАШКА ⟸ u;
                    НЕМАРКИРАН [u] := лъжа;
                    T := T ∪ {v, u}
                    end
        end
end
```

Очевидно сложността на този алгоритъм също е $O(n+m)$.

В дървото (V,T) , построявано от горната процедура, разстоянието от произволен връх v до корена r е най-краткият път от v до r в графа G (тук под дължина на път се разбира броя на участващите в пътя ребра).

По-късно в параграф 2.3 ще бъдат разгледани алгоритми за намиране на най-кратки пътища, като при това на ребрата ще

бъдат съпоставени тегла (дължини) не обезателно равни на 1. В този случай под дължина на пътя ще се разбира сумата от дължините (теглата) на участващите в пътя ребра.

В много случаи, описващи реални проблеми от практиката, не е обезателно графът G да бъде свързан. Тогава се налага да се построяват дървовидни структури (гора от дървета).

2. Алгоритъм за построяване на максимален ориентиран лес (МОЛ)

ПРИМЕР 2.4. Да предположим, че ръководите столична фирма с много клонове, разпръснати из цялата страна. Налага се ежедневно да инструктирате клоновете на фирмата по места. Как да осъществите това, ако разполагате само с телефонни връзки?

Един от начините е да вдигнете телефона и се свържете с всеки клон. Очевидно, това е твърде нерентабилно, както по отношение на пари, така и на време. По-добре е да създадете система, порядък, като укажете на всеки получил съобщението (и то веднъж) на кого трябва да го предаде (телефонира). По този начин например, вместо k разговора с черноморските клонове на фирмата, можете да позвъните само веднъж до Бургас, а бургаският клон да се свърже с останалите от региона.

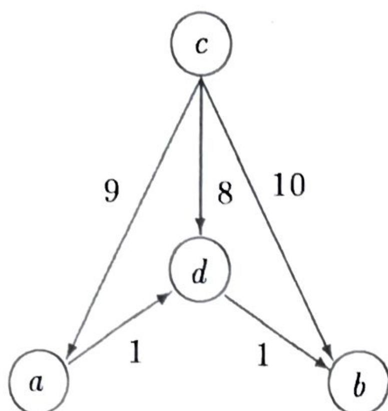
Подобен род задачи налагат необходимостта в даден граф G да се построяват покриващи ориентирани лесове с максимално (минимално) възможни тегла. Ще ги наричаме *максимален (минимален) ориентиран лес* за графа G. За краткост ще ги бележим с **МОЛ (мол)**. Максималните (минималните) ориентирани дървета ще бележим с **МОД (мод)**. Ясно е, че за разлика от досега разгледаните алгоритми в този параграф, вече ще отчитаме и ориентацията на ребрата.

Нека и сега повикаме интуицията на помощ и формулираме следния "*поглъщащ алгоритъм за построяване на* **МОЛ**".

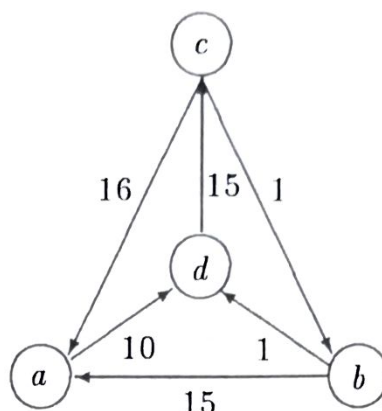
"Подреждаме дъгите по тегла — първа е тази с най-голямо тегло и т.н., а дъгите с еднакви тегла номерираме произволно. Започваме формирането на ориентиран лес от дъгата с най-голямо тегло. Последователно разглеждаме дъгите и включваме в леса тези от тях, които образуват ориентиран лес с разгледаните вече дъги. След изчерпване множеството на дъгите спираме. Полученият ориентиран лес е максимален."

Оказва се, че интуицията в този случай, за разлика отпреди, се явява лош съветник. Този "алгоритъм" работи, но невинаги — с други думи не е алгоритъм.

ПРИМЕР 2.5. Да се приложи горната процедура за търсене на **МОЛ** за следните графи G_1 и G_2:



Граф G_1



Граф G_2

Ако приложим формулирания "алгоритъм" за графа G_1, наистина ще получим **МОЛ**, образуван от дъгите (c, b), (c, a), (c, d) с тегло $10 + 9 + 8 = 27$. (Проверете теглата на всички други лесове в G_1).

За графа G_2 "алгоритъмът" работи лошо. Според предписанията му, лесът (c, a), (d, c), (b, d) или (c, a), (d, c), (c, b) е **МОЛ**, което не е вярно. Например (b, a), (a, d), (d, c) е ориентиран лес с тегло 40, докато генерираният от "алгоритъма" лес е с тегло 32.

Да разгледаме сега *алгоритъма на Едмондс* [14] за построяване на **МОЛ**.

ИДЕЯ НА АЛГОРИТЪМА ЗА ПОСТРОЯВАНЕ НА **МОЛ.**

Алгоритъмът се състои от два основни етапа. В първия се разглеждат последователно върховете на графа и се извършва "свиване" ("стягане", "умаляване") на графа, а във втория се извършва обратното — разтягане (разширяване), докато стигнем до изходния граф и **МОЛ**.

Алгоритъмът използва (поддържа и актуализира) два букета (множества) — букет от върхове V_i и букет от дъги E_i. Букетът от върхове съдържа разгледаните при изпълнението на алгоритъма върхове, а букетът дъги — условно включените в МОЛ дъги, инцидентни с разглежданите върхове. В началото тези два букета са празни множества.

1. В произволен ред се разглеждат и включват в букета V_i върховете на графа, като от влизащите във върха дъги се избира тази с максимално положително тегло. (Върховете без входящи дъги могат да се игнорират от разглеждане). Тази дъга се включва в букета дъги E_i, ако това не нарушава свойството на букета да бъде лес (добавянето ѝ не води до образуване на цикъл). В противен случай върховете и дъгите на цикъла се "свиват" (стягат) в един

връх v_{i-1}. В новия "умален" (свит) граф се променят теглата на някои дъги и за него се променя съставът на двата букета — остават само тези върхове и дъги, които са от новия граф. След корекциите продължава процедурата разглеждане на върховете, докато не бъдат разгледани всички върхове.

2. След като всички върхове на графа са разгледани, дъгите на последния формиран букет образуват лес в съответния граф (получен след неколкократно "свиване", стягане на изходния граф). Когато е извършено поне едно свиване, започва обратен процес на "разширяване" (уголемяване), като всеки фиктивен връх в съответния граф се заменя с цикъла, който преди това е бил "свит" до този връх. В разширения граф и съответния му букет дъги се включват всички дъги на цикъла, с изключение на една, така че новият букет да образува лес. Извършват се толкова разширения, колкото са били свиванията, докато се възстанови изходният граф. При това, дъгите от последния букет дъги образуват **МОЛ**.

Изложените бележки за същността на алгоритъма **МОЛ** не са достатъчно пълни, прецизни и ясни. Ще дадем едно по-формално описание на алгоритъма.

ОПИСАНИЕ НА АЛГОРИТЪМА МОЛ (НАМИРАНЕ НА МАКСИМАЛЕН ОРИЕНТИРАН ЛЕС). Нека изходният граф е G_0, а графите, които се получават в хода на изпълнението на алгоритъма, са $G_1, G_2, \ldots$. Да означим букетите върхове и дъги на тези графи съответно с $V_0, V_1, V_2, \ldots$ и $E_0, E_1, E_2, \ldots$

СТЪПКА 1. Всички букети са празни, $i := 0$.

СТЪПКА 2. Ако всички върхове на G_i са в букета V_i, преминете към *стъпка 4.* В противен случай изберете произволен връх v на графа G_i. Изберете от влизащите във v дъги тази с най-голямо положително тегло и я включете в букета E_i, а върха v — във V_i. Ако такава дъга няма, се върнете в началото на *стъпка 2.*

Проверете дали след включването на дъгата E_i остава лес. Ако това е така (т.е. няма цикли), върнете се в началото на *стъпка 2.* В противен случай преминете към *стъпка 3.*

СТЪПКА 3. Означете появилия се цикъл с ρ_i. Свийте всички дъги и върхове на ρ_i в един връх (фиктивен) v_i. Получения нов граф обозначете с G_{i+1}. Теглата на дъгите в G_{i+1}, невходящи

във v_i, остават каквито са били в G_i. Теглата на дъгите (x, y) от G_i, които в G_{i+1} са дъги от вида (x, v_i) се коригират по следния начин.

$$c(x, v_i) = c(x, y) + c(p, q) - c(r, y), \tag{2.4}$$

където (p, q) е дъгата от контура ρ_i с минимално тегло, а (r, y) е дъгата от контура ρ_i, влизаща във върха y. Очевидно

$$c(r, y) \geq c(p, q) \geq 0 \quad \text{и} \quad c(r, y) \geq c(x, y).$$

Сформирайте съответни за G_{i+1} букети V_{i+1} и E_{i+1}, като от старите букети V_i и E_i вземате само онези върхове и дъги, които са от G_{i+1}.

Увеличете i с единица, т.е. $i := i + 1$ и преминете към *стъпка 2.*

СТЪПКА 4. Ако $i = 0$, край на алгоритъма. Дъгите от букета E_0 образуват максимален ориентиран лес за изходния граф G_0. В противен случай преминете към *стъпка 5.*

СТЪПКА 5. Ако върхът v_{i-1} е корен на някое ориентирано дърво в ориентирания лес E_i: заменете върха v_{i-1} с цикъла ρ_{i-1} и обединете дъгите от E_i с дъгите от цикъла ρ_{i-1}; от това обединение, съдържащо точно един цикъл — ρ_{i-1}, отстранете дъгата от цикъла с минимално тегло и така получения нов букет (лес в графа G_{i-1}) означете с E_{i-1}; $i := i - 1$ и се върнете към *стъпка 4.* В противен случай, т.е. когато v_{i-1} не е корен, преминете към *стъпка 6.*

СТЪПКА 6. На единствената дъга (x, v_{i-1}) от ориентирания лес E_i съответства дъга (x, y) от G_{i-1}, където върхът y е от цикъла ρ_{i-1} (свит във v_{i-1}). Заменете върха v_{i-1} с цикъла ρ_{i-1} и обединете дъгите от E_i с дъгите от цикъла ρ_{i-1}. Полученото множество от дъги съдържа единствен цикъл — ρ_{i-1} и точно две дъги, влизащи във върха y — дъгата (x, y) и съответната дъга от цикъла ρ_{i-1}. Отстранете от обединението дъгата от цикъла ρ_{i-1}, влизаща във върха y и означете получения нов букет с E_{i-1} (ориентиран лес в графа G_{i-1}). Положете $i := i - 1$ и преминете към *стъпка 4.*

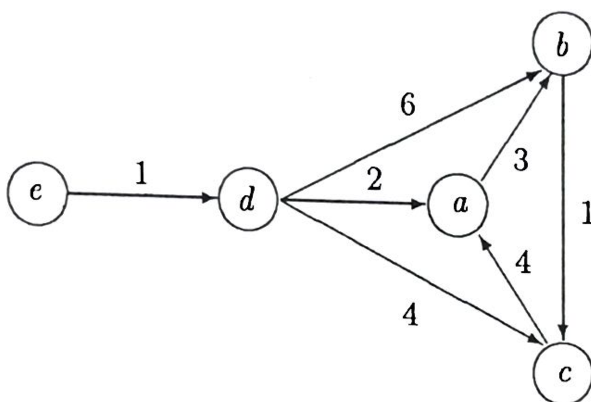
Основната идея при доказателството (обосновката) на алгоритъма се състои в това, че ако ориентираният лес в графа G_k, определен от дъгите в букета E_k е максимален, то такъв се явява и лесът в графа G_{k-1}, определен от дъгите в букета E_{k-1} (вж. [1]).

Ще илюстрираме работата на алгоритъма със следните задачи.

ЗАДАЧА 2.5. Ако всички дъги на графа G са с отрицателни тегла, какъв максимален ориентиран лес генерира алгоритъмът.

Отг.: $\emptyset$ (изпълняват се само стъпки 2 и 4, като $E_0 = \emptyset$).

ЗАДАЧА 2.6. Да се построи максимален ориентиран лес за следния граф G_0



Решение: За прегледност решението е изложено в таблица.

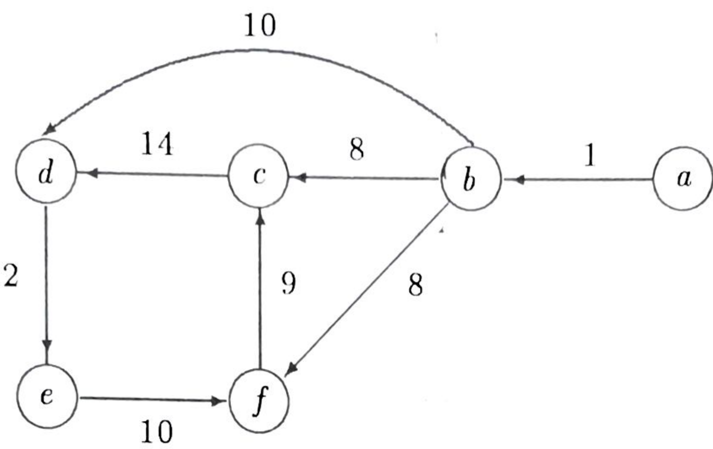
$$i := 0.$$

Връх	V_0 (букет от върхове)	E_0 (букет от дъги)
d	e, d	(e, d)
a	e, d, a	$(e, d), (c, a)$
c	e, d, a, c	$(e, d), (c, a), (d, c)$
b	e, d, a, c, b	$(e, d), (c, a), (d, c), (d, b)$

Поради това, че във V_0 попаднаха всички върхове на графа, а дъгите от букета E_0 образуват лес (стъпки 2 и 4 от алгоритъма), следва, че това е максималният ориентиран лес за графа. Теглото на **МОЛ** е 15. В случая не се наложи да се изпълнява *СТЪПКА 3* — процедурата за свиване на графа.

ЗАДАЧА 2.7. Да се построи максимален ориентиран лес за графа G_0

Граф G_0 :



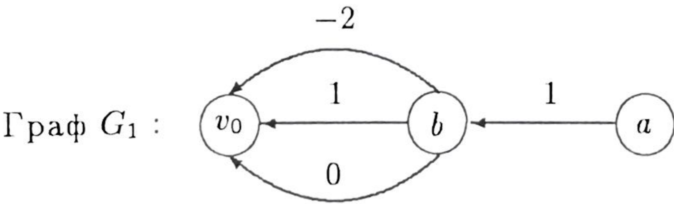
Решение:

$$i := 0.$$

Връх	V_0	E_0
b	b	(a,b)
d	b,d	$(a,b),(c,d)$
e	b,d,e	$(a,b),(c,d),(d,e)$
f	b,d,e,f	$(a,b),(c,d),(d,e),(e,f)$
c	b,d,e,f,c	$(a,b),(c,d),(d,e),(e,f),(f,c)$

След включването на дъгата (f,c) в E_0, това множество от дъги вече не е ориентиран лес. *СТЪПКА 2* на алгоритъма ни препраща към *СТЪПКА 3*. Всички дъги и върхове от цикъла ρ_0 : $(c,d),(d,e),(e,f),(f,c)$ свиваме в един връх — върха v_0. Получаваме граф G_1, който изглежда, както е показано на следващия чертеж, като теглата на дъгите му се пресмятат съгласно инструкциите от *СТЪПКА 3* и формулата (2.4):

Граф G_1 :



Теглото на дъгата $(a,b) = 1$, колкото беше и в графа G_0. Тъй като дъгата с минимално тегло от цикъла е дъгата (d,e), за теглата на входящите във v_0 дъги имаме (формула (2.4))

$$c(b,v_0)_1 = c(b,d) + c(d,e) - c(c,d) = 10 + 2 - 14 = -2,$$

$$c(b,v_0)_2 = c(b,c) + c(d,e) - c(f,c) = 8 + 2 - 9 = 1,$$

$$c(b,v_0)_3 = c(b,f) + c(d,e) - c(e,f) = 8 + 2 - 10 = 0.$$

Сформираме букети V_1 и E_1, както е указано в *СТЪПКА 3*, като от букетите V_0 и E_0 взимаме онези елементи (върхове и дъги), които са в G_1. Увеличаваме i, $i := i + 1$ и преминаваме отново към *СТЪПКА 2*.

Връх	$V_1 = \{b\}$	$E_1 = \{(a, b)\}$
–	b	(a, b)
v_0	b, v_0	$(a, b), (b, v_0)_2$
a	b, v_0, a	$(a, b), (b, v_0)_2$

Тъй като в букета V_1 попадат всички върхове на графа G_1 и дъгите от E_1 — $(a, b), (b, v_0)_2$ образуват ориентиран лес, *СТЪПКИ 1, 4 и 5* ни препращат до *СТЪПКА 6*. Съгласно предписанията на *СТЪПКА 6* заменяме върха v_0 с цикъла ρ_0 и обединяваме дъгите от $E_1 = \{(a, b), (b, v_0)_2\}$ с дъгите от цикъла ρ_0 : $(c, d), (d, e), (e, f), (f, c)$. Получаваме множеството дъги $(a, b), (b, c), (c, d), (d, e), (e, f), (f, c)$, от което отстраняваме дъгата от цикъла, влизаща във върха c, т.е. дъгата (f, c). Така полученото множество от дъги е новия букет E_{i-1}, т.е. E_0

$$E_0 = \{(a, b), (b, c), (c, d), (d, e), (e, f)\}.$$

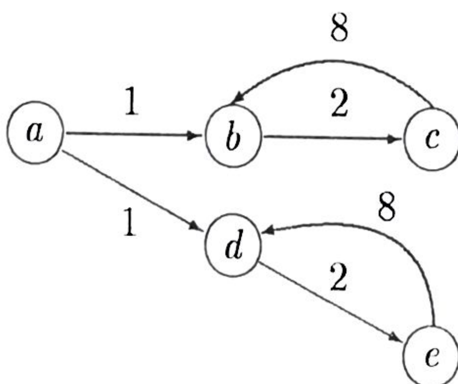
СТЪПКА 6 ни препраща към *СТЪПКА 4* (преди това $i := i - 1$), според която полученият нов букет E_0 е максимален ориентиран лес за изходния граф G_0.

Теглото на намерения лес е $1 + 8 + 14 + 2 + 10 = 35$, и то е възможно максималното (проверете).

В задача 2.7 максималният лес е покриващ. Обърнете внимание на следните забележки.

1. Максималният ориентиран лес (максималното ориентирано дърво) не се явява задължително покриващ лес (покриващо ориентирано дърво).

   Пример: В графа G, изобразен на черт. 2.1,



Черт. 2.1

максималният ориентиран лес е $\{(c, b), (e, d)\}$ и теглото му е 16. Този лес не е покриващ лес, не е и покриващо дърво (макар, че в графа съществува покриващо дърво и всички тегла са положителни).

2. Покриващият ориентиран лес (дърво) невинаги е максимален(о).

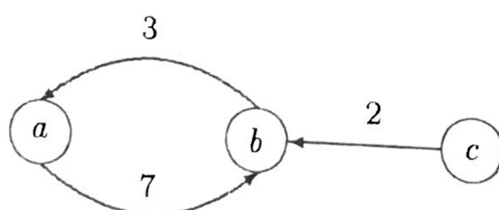
   За графа от черт. 2.1 лесът $(c, b), (a, d), (d, e)$ е покриващ с тегло 11 и очевидно не е максимален $(11 < 16)$.

   Покриващото дърво за графа от черт. 2.1, $(a, b), (b, c), (a, d)$, (d, e) очевидно не е максимално, тъй като теглото му е 6, а максималното дърво е с тегло 8 (в случая за всяко покриващо дърво е изпълнено 2).

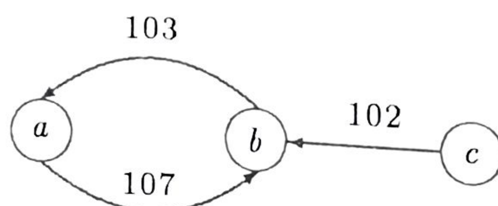
3. Никой ориентиран лес не може да има повече дъги от покриващото ориентирано дърво (когато такова дърво съществува в графа).

Нека теглата на дъгите на произволен граф G считаме за равни — примерно всички тегла са единица. Тогава максималният ориентиран лес ще бъде лес с максимален брой дъги и поради 3., алгоритъмът за търсене на **МОЛ** в такъв граф ще генерира покриващо ориентирано дърво (когато такова съществува). Нека към всички тегла добавяме една положителна константа M. С нарастването на M ще расте и броят на дъгите в **МОЛ**. Поради 3., ако M е достатъчно голямо, алгоритъмът за търсене на **МОЛ**, приложен към граф с така променени тегла, ще генерира покриващо ориентирано дърво с максимално тегло (когато такова съществува).

ПРИМЕР 2.6. Да разгледаме следния граф.



Прилагането на алгоритъма **МОЛ** за този граф генерира максимален лес $\{(a, b)\}$ с тегло 7. Ако всички тегла увеличим със 100 и приложим същия алгоритъм за графа,



ще получим максимален ориентиран лес $\{(c,b),(b,a)\}$ с тегло 205, който се явява покриващо дърво с максимално тегло за първия от двата графа.

Ще отбележим, че алгоритъмът за търсене на **МОЛ** може да се използва за:

а) търсене на *минимален ориентиран лес* (**мол**). За целта е достатъчно да умножите с (-1) теглата на всички дъги на изходния граф.

   Алгоритъмът за търсене на **МОЛ**, приложен за графа с новите тегла, очевидно ще генерира лес, който е минимален ориентиран лес за изходния граф.

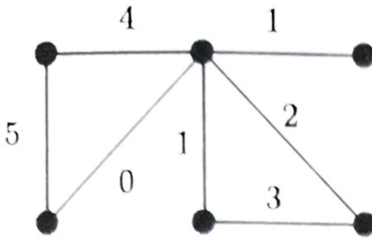
б) търсене на *максимално покриващо ориентирано дърво*. За целта към теглата на всички дъги се добавя достатъчно голяма, положителна константа M. Както вече беше отбелязано, алгоритъмът за търсене на **МОЛ**, приложен за графа с променени тегла, ще генерира лес, който е покриващо ориентирано дърво с максимално тегло (когато покриващо дърво съществува).

в) търсене на *минимално покриващо ориентирано дърво* (когато съществува). За целта умножете всички тегла с (-1) и добавете достатъчно голяма положителна константа M към получените тегла. За графа, с така променени тегла, приложете алгоритъма **МОЛ**.

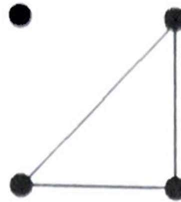
г) търсене на *максимално (или минимално) покриващо ориентирано дърво с корен в даден връх x* (когато това е възможно). За целта в графа се добавя допълнителен връх y и дъгата (y,x) с произволно тегло. Ако съществува покриващо ориентирано дърво в разширения граф, то ще има за корен върха y, тъй като в y няма влизащи дъги. Намереното покриващо ориентирано дърво за разширения граф, след отстраняване на дъгата (y,x) и върха y, става такова дърво и за изходния граф. Следователно, за целта е достатъчно в разширения с дъгата (y,x) граф да променим теглата, както в т. б) и в) и приложим алгоритъма за **МОЛ**.

Сега вече е ясно, че проблемът, поставен в пример 2.4, се свежда до търсене на минимално покриващо ориентирано дърво с корен във върха, съответен на град София.

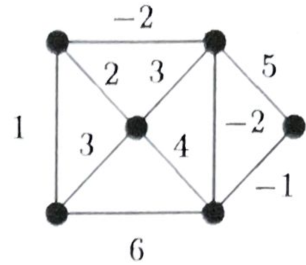
ЗАДАЧА 2.8. Да се построят всички покриващи дървета за всеки от следните графи. Съставете компютърна програма за търсене на покриващи дървета.



Граф G_1

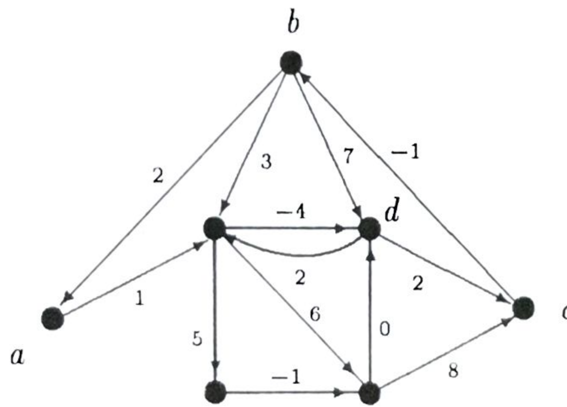


Граф G_2



Граф G_3

ЗАДАЧА 2.9. Да се построят максимално и минимално покриващо дърво за графите от задача 2.8, без да се търсят всички покриващи дървета.



Граф G_4

ЗАДАЧА 2.10. За графа G_4 да се построят

а) максимален ориентиран лес;

б) минимален ориентиран лес;

в) максимално покриващо ориентирано дърво;

г) минимално покриващо ориентирано дърво;

д) максимално (минимално) покриващо ориентирано дърво с корени съответно във върховете a, b, c, d.

ЗАДАЧА 2.11. Формулирайте реални задачи (икономически, управленски, инженерни и т.н.), определете техния модел на езика на графите и съставете компютърни програми за тяхното решаване.

2.3. Алгоритми за търсене на пътища

ПРИМЕР 3.1. Да разгледаме граф $G = (V, E)$, върховете на който съответстват на летищата в света, а дъгите — на въздушните линии между тях (възможните полети). Как да се избере маршрут за пътуване между два пункта, така че този маршрут (път) да бъде оптимален в някакъв смисъл?

Оптималността може да бъде свързана с дължина, бързина, цена, сигурност, комфорт и др.

Нека на всяка дъга на графа е съпоставено тегло $c(x, y)$, което се интерпретира като километри, време, печалба и т.н. Тези тегла могат да се зададат и в матрица $C = (c_{ij})$. Елементите на *матрицата на теглата* могат да са положителни, отрицателни или нула. Защо теглата могат да са и отрицателни?

ПРИМЕР 3.2. Търговски пътник предвижда пътуване от София до Варна, като възнамерява пътьом да посети и няколко други градове. Търговецът с голяма точност знае каква печалба ще му донесе евентуалното посещение на клиенти в съответния град. Какъв маршрут да избере този търговец?

Да разгледаме граф G с върхове градовете, които е възможно да посети търговецът.

Ако теглото на всяка дъга означава

"разходи за път" — "очаквани приходи",

е ясно, че е възможно да съществуват дъги с отрицателни тегла (участъци, където транспортните разходи са по-малки от очакваната печалба), както и дъги с положителни и нулеви тегла. В този случай търговският пътник трябва да си избере маршрут, съответстващ на най-краткия път в графа между върховете "София" и "Варна".

Ще отбележим, че алгоритмите за търсене на *най-кратък път* (**НКП**) са различни в случаите "всички тегла са неотрицателни" и "теглата са произволни".

Освен това, при отсъствието на една дъга (x, y) от графа ще считаме, че теглото ѝ е $c(x, y) = \infty$. Теглата на дъгите и пътищата ще наричаме с най-естествения за случая термин — *дължини*.

В задачите за намиране на **НКП** (*най-кратък път*) се налага ограничението в графа G да няма цикли с отрицателно тегло. Ясно е, че наличието на такива цикли прави несъдържателна задачата, поради възможността цикълът да се обхожда многократно и пътят между два върха да става безкрайно малък (да клони към $-\infty$). За елементите c_{ij} от матрицата на теглата ще предполагаме, че не е задължително да удовлетворяват условието на триъгълника

$$c_{ij} \le c_{ik} + c_{kj}, \quad \text{за } \forall i,\ j,\ k,$$

поради тривиалността на задачата в противен случай.

В този параграф ще разгледаме и алгоритъм за търсене на *втори, трети,..., k-ти по дължина пътища*, което често се налага и е полезно при решаването на многокритериални задачи от практиката.

Поради това, че теглото на един път невинаги се явява сума от теглата на участващите в него дъги, в този параграф ще разгледаме и въпроса, свързан с намирането на пътища с максимална надеждност и пропускателна способност.

Такива задачи могат да се преформулират като задачи за **НКП** или да се приспособят за решаването им методите, използвани в задачите за намиране на **НКП**.

1. Най-кратък път (НКП) между два дадени върха s и t. Алгоритъм на Дийкстра

Алгоритъмът е предложен от *Dijkstra E., 1959 g.*, [15] и е твърде ефективен и простичък. В този алгоритъм се предполага, че *дължините c_{ij} на всички дъги са неотрицателни.*

ИДЕЯ НА АЛГОРИТЪМА. Алгоритъмът може да се разглежда като процес на последователно маркиране върховете на графа със съответни числа. В общия случай *маркиращото число $d(x)$* на върха x е временно и дава горна граница за дължината на пътя от s до x. При изпълнението на алгоритъма стойностите на маркиращите числа се намаляват, като на всяка итерация точно едно от временните маркиращи числа става *постоянно (оцветява се)*. В този случай *постоянното (оцветеното)* маркиращо число $d(x)$ вече не се явява някаква горна граница, а е точната дължина на **НКП** от s до x. За оцветен считаме и съответния връх x. Когато освен дължината на **НКП** се търси и самият път, се оцветява и една от дъгите на графа, като по този начин тя се включва в търсения път.

ОПИСАНИЕ АЛГОРИТЪМА НА ДИЙКСТРА. *СТЪПКА 1.* Оцветете началния връх s, положете

$d(s) = 0$ (постоянно маркиращо число),
$d(x) = \infty$ (временни маркиращи числа), $\forall x \neq s$,
$p = s$ (p — последния оцветен връх).

СТЪПКА 2. (*Промяна на временните маркиращи числа.*) За

всички неоцветени върхове x преизчислете числата $d(x)$ по формулата

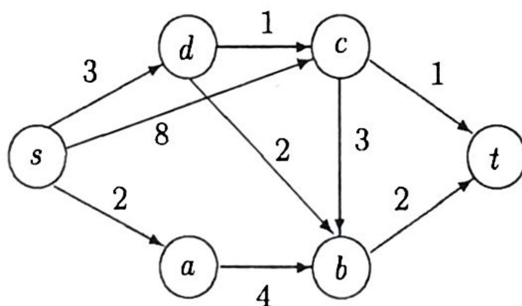
$$(3.1) \qquad d(x) = \min\{d(x),\ d(p) + c(p,x)\}$$

(Очевидно се променят само онези $d(x)$, за които съществува дъга (p, x), останалите маркиращи числа остават същите.)

Ако за всеки неоцветен връх x, $d(x) = \infty$, прекратете процедурата — в графа няма пътища от s до неоцветените върхове. В противен случай оцветете онзи връх x, чието число $d(x)$ е минимално. Оцветете и дъгата, влизаща в x, за която се достига минимума от (3.1). Положете $p = x$.

СТЪПКА 3. Ако $p = t$, край на процедурата, единственият път от s до t, съставен на оцветени дъги, е **НКП** между s и t. В противен случай преминете към *стъпка 2.*

ПРИМЕР 3.3. Да се намери с алгоритъма на Дийкстра **НКП** между s и t в следния граф.



Да припомним, че при липса на дъга (u, v), теглото ѝ считаме за ∞.

СТЪПКА 1. Полагаме $d(s) = 0$ и $d(x) = \infty$, за всички други върхове. Оцветяваме s (няма дъги за оцветяване). Полагаме $p = s$.

СТЪПКА 2. По формула (3.1) пресмятаме новите маркиращи числа за неоцветените върхове на графа.

$$d(a) = \min\{d(a),\ d(s) + c(s,a)\} = \min\{\infty, 0+2\} = 2,$$

$$d(d) = \min\{d(d),\ d(s) + c(s,d)\} = \min\{\infty, 0+3\} = 3,$$

$$d(c) = \min\{d(c),\ d(s) + c(s,c)\} = \min\{\infty, 0+8\} = 8.$$

Излишно е да преизчисляваме останалите маркиращи числа, тъй като остават непроменени — в случая ∞ (проверете).

Тъй като $d(a)$ е минималното маркиращо число измежду преизчислените числа, *оцветяваме върха a и дъгата* (s, a). Полагаме $p = a$.

СТЪПКА 3. Препраща ни към *стъпка 2*, тъй като върхът t не е оцветен.

СТЪПКА 2. $(p = a)$

$$d(b) = \min\{\infty,\ d(a) + c(a,b)\} = \min\{\infty, 2+4\} = 6,$$

$$d(d) = 3,\ d(c) = 8,\ d(t) = \infty.$$

Минимално е числото $d(d) = 3$, следователно *оцветяваме върха d и дъгата (s, d)*. Полагаме $p = d$.

СТЪПКА 3. Препраща ни към *стъпка 2.*

СТЪПКА 2. $(p = d)$

$$d(c) = \min\{8,\ d(d) + c(d, c)\} = \min\{8, 3 + 1\} = 4,$$

$$d(b) = \min\{6,\ d(d) + c(d, b)\} = \min\{6, 3 + 2\} = 5,$$

$$d(t) = \infty.$$

Минималното от временните маркиращи числа (тези на неоцветените върхове) е $d(c) = 4$. *Оцветяваме върха c и дъгата (d, c).* Полагаме $p = c$.

СТЪПКА 3. Препраща ни към *стъпка 2.*

СТЪПКА 2. $(p = c)$

$$d(b) = \min\{5, 4 + 3\} = 5,$$

$$d(t) = \min\{\infty,\ d(c) + c(c, t)\} = \min\{\infty, 4 + 1\} = 5.$$

Минимално е всяко от числата $d(b) = d(t) = 5$. Ако изберем $d(t)$, ще *оцветим върха t и дъгата (c, t)*. Полагаме $p = t$.

СТЪПКА 3. Край на алгоритъма. Построеното дърво (s, a), (s, d), (d, c), (c, t) на най-кратките пътища, дава и най-краткия $(s - t)$ път

$$(s, d),\ (d, c),\ (c, t),$$

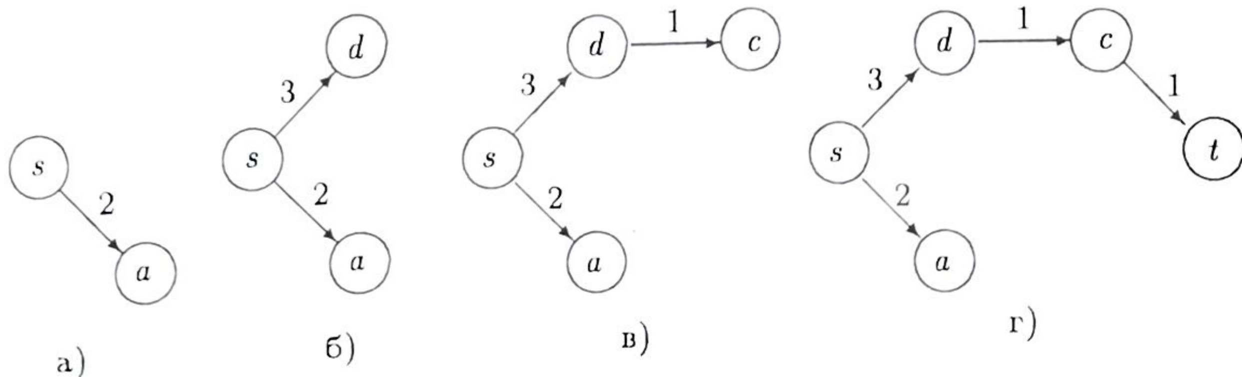
който е с дължина $3 + 1 + 1 = 5$.

Ако бяхме избрали за минимално не $d(t) = 5$, а $d(b) = 5$, вместо върха t и дъгата (c, t), трябваше *да оцветим върха b и дъгата (d, b)*. Полагаме $p = b$. *СТЪПКА 3* ни препраща отново към *СТЪПКА 2* $(p = b)$. Оцветяваме *върха t и дъгата (c, t)*. Край на алгоритъма. Построеното дърво (s, a), (s, d), (d, c), (d, b), (c, t) дава **НКП** от s до всеки връх на графа, в това число и $(s - t)$ **НКП**, а именно

$$(s, d),\ (d, c),\ (c, t),$$

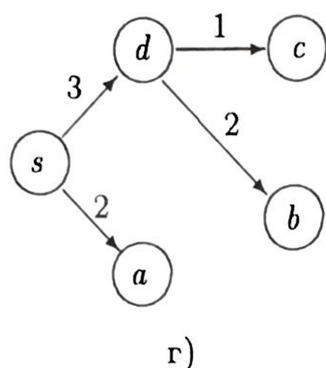
чиято дължина е $3 + 1 + 1 = 5$.

Графично, етапите на строене на пътя може да се илюстрират, както е показано по-долу.

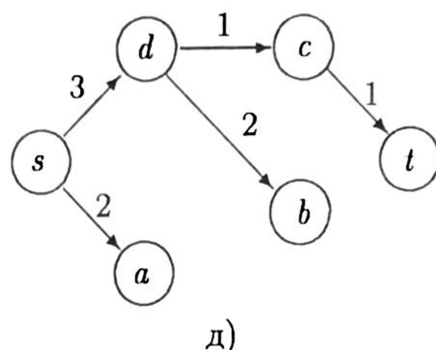


а) б) в) г)

или при втория избор, когато $d(b)$ приемем за минимално вместо епапа г), имаме:



г)



д)

КОМЕНТАР И ОБОСНОВКА НА АЛГОРИТЪМА. Всеки път, когато се оцветява връх (без началния s), се оцветява и точно една дъга, влизаща в него. Следователно, при изпълнението на алгоритъма в един връх не може да влиза повече от една оцветена дъга. Оцветените дъги (строеният от алгоритъма път) не образуват контур, тъй като алгоритъмът не допуска оцветяването на дъга, чийто краища (върхове) вече са оцветени. От казаното следва, че алгоритъмът строи в изходния граф ориентирано дърво с корен във върха s, състоящо се от оцветени дъги. То се нарича *ориентирано дърво на най-кратките пътища*. Единственият път от s до всеки връх x на това дърво се явява най-краткият път между върховете s и x. Очевидно, ако имаме $(s - x)$ **НКП** и върхът y принадлежи на този път, то частта от пътя между y и x се явява най-краткият път между върховете y и x. (Допускането на противното води до противоречие с минималността на $(s - x)$ пътя.)

Формулираният алгоритъм много лесно се модифицира, за да търси не само **НКП** между s и t, а между s и всички останали върхове на графа, т.е. да построява покриващо ориентирано дърво с корен s. За целта в *СТЪПКА 3*, критерият не трябва да бъде дали е оцветен върхът t, а дали са оцветени всички върхове на графа (при условие, че в графа се съдържа поне едно покриващо ориентирано дърво).

Малко по-късно ще дадем обосновка на една модификация на този алгоритъм, когато разгледаме алгоритъма на Форд. При машинна реализация на алгоритъма на Дийкстра, след получаване дължините на най-кратките пътища от s до останалите върхове, самите пътища могат да се получат с рекурсивна процедура, използвайки формулата

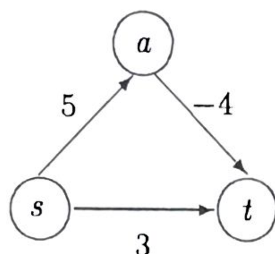
$$d(x_i) = d(x_i') + c(x_i', x_i), \tag{3.2}$$

където x_i' е върхът, непосредствено предхождащ x_i в **НКП**.

При графи с неориентирани дъги, т.е. ребра, всяко ребро може да се разглежда като двойка противоположно ориентирани дъги с равни тегла.

Очевидно $(s-t)$ **НКП** ще бъде единствен, когато при изпълнение на алгоритъма нито веднъж не възниква нееднозначност при избора на дъга за оцветяване.

ЗАДАЧА 3.1. Да се приложи алгоритъмът на Дийкстра за намиране на $(s-t)$ **НКП** в следния граф.



Решение: $d(s)=0$, $d(t)=\infty$, $d(a)=\infty$. Оцветяваме върха s и полагаме $p=s$. Преизчисляваме $d(t)$ и $d(a)$.

$$d(a)=\min\{d(a),\ d(s)+c(s,a)\}=\min\{\infty,0+5\}=5,$$

$$d(t)=\min\{d(t),\ d(s)+c(s,t)\}=\min\{\infty,0+3\}=3.$$

Минимално е $d(t)$, следователно оцветяваме върха t и дъгата (s,t). Край на алгоритъма. Пътят (s,t) е "**НКП**", дължината му е 3. Това очевидно не е вярно, тъй като пътят (s,a), (a,t) е с дължина $5-4=1$ и той е по-кратък.

И така, от задача 3.1 се вижда, че алгоритъмът на Дийкстра не работи добре (т.е. е неприложим) в случаите, когато теглата (дължините) на дъгите могат да са и отрицателни числа.

2. Алгоритъм на Форд за НКП [16]

Този алгоритъм е модификация на алгоритъма на Дийкстра за случаите, когато някои дъги имат отрицателни тегла. Разликите се състоят в следното.

а) В *СТЪПКА 2* на алгоритъма на Форд, по формула (3.1) се преизчисляват числата $d(x)$ не само за неоцветените върхове, а за всички върхове;

б) Ако за оцветен връх x се получи намаляване стойността на маркиращото число $d(x)$, то оцветяването на върха x и инцидентната с него оцветена дъга се снема (игнорира);

в) Алгоритъмът спира своята работа, когато всички върхове са оцветени и след изпълнение на *СТЪПКА 2* никое от числата $d(x)$ не се променя.

ЗАДАЧА 3.2. Да се приложи алгоритъмът на Форд за намиране $(s-t)$ **НКП** в графа от задача 3.1.

Решение: $d(s)=0$, $d(a)=\infty$, $d(t)=\infty$. Оцветяваме върха s, полагаме $p=s$ и преизчисляваме по (3.1).

$$d(a)=\min\{d(a),\ d(s)+c(s,a)\}=\min\{\infty,0+5\}=5,$$

$$d(t)=\min\{d(t),\ d(s)+c(s,t)\}=\min\{\infty,0+3\}=3.$$

Минимално е $d(t)$ — *оцветяваме върха t и дъгата* (s,t). Полагаме $p=t$. Не са оцветени всички върхове, повтаряме *СТЪПКА 2.*

Тъй като от t не излизат дъги, всички маркиращи числа остават същите $d(s)=0$, $d(a)=5$ и $d(t)=3$. *Оцветяваме върха a и дъгата* (s,a). Полагаме $p=a$. Дървото на **НКП** се състои вече от дъгите (s,t) и (s,a). Връщаме се отново към *СТЪПКА 2*, за да проверим променят ли се маркировките на върховете.

$$d(t)=\min\{d(t),\ d(a)+c(a,t)\}=\min\{3,5+(-4)\}=1,$$

$$d(s)=\min\{d(s),\ d(a)+c(a,s)\}=\min\{0,5+\infty\}=0.$$

Тъй като $d(t)$ се намали от 3 на 1, *снемаме оцветяването на върха t и дъгата* (s,t). Сега вече дървото на **НКП** се състои само от дъгата (s,a), като единственият неоцветен връх в графа е t. *Оцветяваме t и дъгата* (a,t). Полагаме $p=t$. Тъй като от t не излизат дъги, величините $d(x)$ не се променят, а няма и неоцветени върхове. Край.

Най-краткият $(s-t)$ път е от оцветените дъги (s,a), (a,t) и дължината му е $5-4=1$.

ОБОСНОВКА НА АЛГОРИТЪМА НА ФОРД. Нека процедурата на алгоритъма на Форд е завършена. Тогава очевидно за произволни x и y ще бъде изпълнено

$$d(x)+c(x,y)\geq d(y), \tag{3.3}$$

тъй като всички върхове са оцветени и маркиращите числа не се променят (отрицанието на (3.3) води до снемане оцветяването на върха y).

Да допуснем, че алгоритъмът е приключил своята работа и за някой връх y на графа, $d(y)$ не съвпада с дължината на най-краткия път от s до y. Ако има няколко такива върха y, избира се този връх y, за който $(s-y)$ **НКП** има най-малък брой дъги.

Да означим с x предпоследния връх в $(s-y)$ **НКП**. При положение, че има няколко такива, за предпоследен връх x се взима онзи, за който $(s-x)$ **НКП** е с най-малък брой дъги. Генерираните от алгоритъма числа $d(y)$ задължително са дължини на $(s-y)$ пътища в графа. Допускането, че $d(y)$ не е дължина на най-краткия път води до

$$d(y) > d(x) + c(x, y) \tag{3.4}$$

(като имаме предвид и избора на x и y).

Поради (3.3) и (3.4) отхвърляме допускането, с което обосновката е завършена.

Както беше споменато в началото на параграфа, разглеждат се графи без цикли с отрицателно тегло. При липса на информация за съществуването на такива цикли, алгоритъмът на Форд също е приложим, но в процеса на работа трябва да се "брои" колко пъти се оцветява всеки връх. Ако броят на оцветяванията на някой връх стане n (броя на върховете на графа), се стопира работата на алгоритъма — в графа има контур с отрицателна дължина.

В противен случай, алгоритъмът след краен брой стъпки завършва работата си и дава **НКП**. Това наистина е така по следните причини. Нека в графа няма цикли с отрицателна дължина. Тогава щом за върха x, $d(x)$ стане окончателно, в най-лошия случай всеки от оставащите върхове може отново (или за пръв път) да бъде оцветен само веднъж, преди окончателното оцветяване на един от тези върхове. Следователно никой връх не може да бъде оцветяван повече от $n-1$ пъти.

3. Алгоритми за търсене на всички НКП. Алгоритми на Флойд и Данциг

Ще бъдат разгледани два сходни алгоритъма за търсене на **НКП** между всяка двойка върхове в графа — *алгоритъм на Флойд* [17] и *алгоритъм на Данциг* [18] . Както при предишните алгоритми, за дължините ще са допустими и отрицателни стойности, като ще предполагаме отсъствието на цикли с отрицателна дължина (впоследствие ще бъде вдигната и тази забрана).

Нека върховете на графа са номерирани с естествените числа от 1 до n. Да означим с d_{ij}^m дължината на **НКП** между i-тия и j-тия връх, в който път за междинни върхове се използват само някои от първите m върха на графа. Ако такъв път не съществува, ще считаме $d_{ij}^m = \infty$. Ясно е, че d_{ij}^0 се явява дължината

на най-късата дъга, съединяваща i и j. За всяко i, полагаме $d_{ii}^0 = 0$.

Матрицата с размери $n \times n$, чийто елементи (i, j) съвпадат с d_{ij}^m да означим с D^m. Очевидно за изходния граф веднага се определя матрицата D^0.

ИДЕЯ НА АЛГОРИТЪМА (НА ФЛОЙД). От началната матрица D^0 се определя матрицата D^1, от нея D^2 и т.н. от D^{n-1} се определя матрицата D^n, даваща всички **НКП** в графа. Същината на алгоритъма е следната. Нека са известни:

а) **НКП** между връх i и връх m с допустими междинни върхове измежду първите $(m-1)$ върха;

б) **НКП** между връх m и връх j с допустими междинни върхове измежду първите $(m-1)$ върха;

в) **НКП** между връх i и връх j с допустими междинни върхове измежду първите $(m-1)$ върха.

Тогава **НКП** между i и j с допустими междинни върхове измежду първите m върха, се явява пътят в) или пътят, получен от обединението на пътищата а) и б), т.е.

$$d_{ij}^m = \min\{d_{ij}^{m-1},\ d_{im}^{m-1} + d_{mj}^{m-1}\}.$$

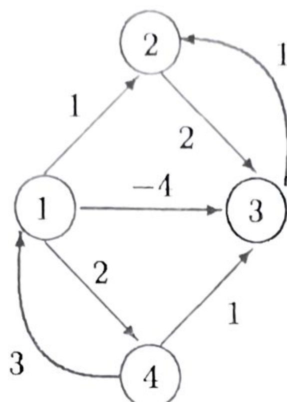
ОПИСАНИЕ НА АЛГОРИТЪМА (НА ФЛОЙД). *СТЪПКА 1.* Всички върхове се номерират с числата от 1 до n. Определя се матрицата $D^0 = (d_{ij}^0)_{n\times n}$, където елементът (i, j) е дължината на най-кратката (с най-малко тегло) дъга между i и j. При липса на дъга (i, j), се полага $d_{ij}^0 = \infty$, а за всяко i, $d_{ii}^0 = 0$.

СТЪПКА 2. За всяко $m \in [1, n]$ се определят елементите на матрицата $D^m = (d_{ij}^m)_{n\times n}$ чрез елементите на матрицата $D^{m-1} = (d_{ij}^{m-1})_{n\times n}$ при използване на рекурсивното съотношение

$$d_{ij}^m = \min\{d_{ij}^{m-1},\ d_{im}^{m-1} + d_{mj}^{m-1}\}. \tag{3.5}$$

Всеки елемент (i, j) на последната матрица D^n задава най-краткия път от върха i до върха j.

ЗАДАЧА 3.3. Да се определи с алгоритъма на Флойд **НКП** между върховете на следния граф.



Решение: В този граф няма цикъл с отрицателно тегло. Последователно пресмятаме елементите на матриците D^1, D^2, D^3 и D^4 (*СТЪПКА 2*), като преди това определяме елементите на D^0 (*СТЪПКА 1*).

$$D^0 = \begin{pmatrix} 0 & 1 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & \infty & 1 & 0 \end{pmatrix}, \quad D^1 = \begin{pmatrix} 0 & 1 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & 4 & -1 & 0 \end{pmatrix} = D^2,$$

$$D^3 = \begin{pmatrix} 0 & -3 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & 0 & -1 & 0 \end{pmatrix}, \quad D^4 = \begin{pmatrix} 0 & -3 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & 0 & -1 & 0 \end{pmatrix}.$$

КОМЕНТАР И ОБОСНОВКА НА АЛГОРИТЪМА НА ФЛОЙД.

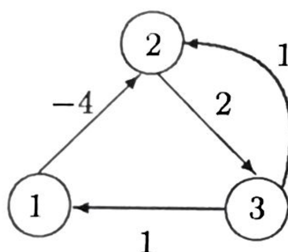
1. Тъй като няма цикли с отрицателни тегла и поради (3.5), то
$$d_{ii}^m = 0, \quad \text{за} \quad \forall i \forall m$$
$$d_{im}^{m-1} = d_{im}^m \quad \text{и} \quad d_{mi}^{m-1} = d_{mi}^m, \ \forall i = 1,\ 2, ...,\ n,$$
т.е. диагоналните елементи могат да не се изчисляват и освен това при определяне елементите на D^m, тези от m-ти ред и m-ти стълб могат да се вземат без изменение от предишната матрица D^{m-1}.

2. Нека липсва информация за съществуването на цикли с отрицателно тегло. Тогава с малка модификация алгоритъмът на Флойд може да се използва за намиране на **НКП** или установяване наличието на цикли с отрицателно тегло. За целта е достатъчно за $\forall i = 1,\ 2, ...$ *да се пресмятат*

d_{ii}^m. Ако $d_{ii}^m < 0$, то в G има цикъл с отрицателно тегло, съдържащ върха x_i. Край. В противен случай, алгоритъмът спира, когато пресметне елементите на последната матрица D^n.

3. Дължината на **НКП** от i до j с допустими междинни върхове измежду първите m върха очевидно не надхвърля дължината на **НКП** от i до j, ако за междинни се използват само първите $(m-1)$ върха и не надхвърля **НКП** от i до j ако за междинни се използват някои от първите $(m-1)$ върха и задължително върха m. Оттук по индукция следва верността на твърдението "алгоритъмът на Флойд генерира **НКП** между всички върхове на графа".

ЗАДАЧА 3.4. Да се определят с алгоритъма на Флойд **НКП** между върховете на следния граф.



Решение: Прилагаме алгоритъма на Флойд, като вземем предвид втората от по-горе направените забележки в коментара, т.е. изчисляваме d_{ii}^m, при $m = 1, 2, \ldots$, и спираме при получаване на $d_{ii}^m < 0$, което индикира наличието на цикъл с отрицателна дължина.

$$D^0 = \begin{pmatrix} 0 & -4 & \infty \\ \infty & 0 & 2 \\ 1 & 1 & 0 \end{pmatrix}, \; D^1 = \begin{pmatrix} 0 & -4 & \infty \\ \infty & 0 & 2 \\ 1 & -3 & 0 \end{pmatrix}, \; D^2 = \begin{pmatrix} 0 & -4 & -2 \\ \infty & 0 & 2 \\ 1 & -3 & -1 \end{pmatrix}.$$

Тъй като $d_{33}^2 = -1$, спираме. В графа има цикъл с отрицателно тегло. Продължаването на процедурата е несъдържателно, тъй като дължините на "НКП" непрекъснато намаляват.

Ще направим още една бележка към коментара на разглеждания алгоритъм.

4. Как ефективно да се определи съставът на дъгите, участващи в най-кратките пътища?

Може да се използва следната техника за съхраняване на информация за самите пътища (а не само за дължините им). Наред с матрицата D^m се поддържа и обновява втора $(n \times n)$ матрица $\Theta^m = (\theta_{ij}^m)$. В началото на матрицата Θ^0 се присвояват

значения $\theta_{ij}^0 = x_i$. В съответствие с (3.5), при изпълнение на *СТЪПКА 2* от алгоритъма, се изчисляват (актуализират) и елементите θ_{ij}^m по следния начин

$$(3.6) \qquad \theta_{ij}^m = \begin{cases} \theta_{ij}^{m-1}, & \text{при } d_{ij}^{m-1} \le d_{im}^{m-1} + d_{mj}^{m-1}, \\ \theta_{mj}^{m-1}, & \text{при } d_{ij}^{m-1} > d_{im}^{m-1} + d_{mj}^{m-1}. \end{cases}$$

В края на алгоритъма най кратките пътища се получават непосредствено от последната матрица Θ^n. Елементът θ_{ij}^n указва върха, непосредствено предхождащ върха x_j в най-краткия път от x_i до x_j. **НКП** между върховете x_i и x_j се дава от следната последователност върхове

$$x_i,\ x_\nu, \dots,\ x_\gamma,\ x_\beta,\ x_\alpha,\ x_j,$$

където

$$\theta_{ij}^n = x_\alpha,\ \theta_{i\alpha}^n = x_\beta,\ \theta_{i\beta}^n = x_\gamma, \dots,\ \theta_{i\nu}^n = x_i.$$

Матрицата Θ^m дава възможност да се определят и дъгите на цикъла с отрицателно тегло, когато $d_{ii}^m < 0$ при някое m.

Този метод се нарича *вътрешен (вграден) метод* и той се използва, когато предварително се знае, че потребителят изисква не само дължините, а и самите **НКП** (*tentative method*).

ПРИМЕР 3.4. За графа от задача 3.3 (вж. решението) да се определят съответните матрици Θ^0, Θ^1, Θ^2, Θ^3, Θ^4.

Решение:

$$D^0 = \begin{pmatrix} 0 & 1 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & \infty & 1 & 0 \end{pmatrix}, \qquad \Theta^0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix},$$

$$D^1 = D^2 = \begin{pmatrix} 0 & 1 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & 4 & -1 & 0 \end{pmatrix}, \quad \Theta^1 = \Theta^2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 1 & 4 \end{pmatrix},$$

$$D^3 = D^4 = \begin{pmatrix} 0 & -3 & -4 & 2 \\ \infty & 0 & 2 & \infty \\ \infty & 1 & 0 & \infty \\ 3 & 0 & -1 & 0 \end{pmatrix}, \quad \Theta^3 = \Theta^4 = \begin{pmatrix} 1 & 3 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 3 & 1 & 4 \end{pmatrix}.$$

Както беше отбелязано в коментара на алгоритъма (вж. 4), елементите на матрицата Θ^4 дават номерата на предхождащите върхове във всеки най-кратък път. Например, да определим най-краткия път между връх 4 и връх 2.

Елементът d_{42}^4 от D^4 дава дължината на най-краткия път между връх 4 и връх 2. Елементът $\theta_{42}^4 = 3$, т.е. в този път, предхождащ за връх 2 е връх 3. Елементът $\theta_{43}^4 = 1$, т.е. следващият предхождащ е връх 1. Елементът $\theta_{41}^4 = 4$ — край, стигнахме до началния връх на пътя. Следователно, самия най-кратък път (който е с дължина $d_{42}^4 = 0$) е $(4,1)$, $(1,3)$, $(3,2)$. Анализирайте и останалите елементи на матрицата Θ^4, т.е. определете най-кратките пътища между всеки два върха с помощта на тази матрица.

ЗАДАЧА 3.5. Да се намерят матриците Θ^0, Θ^1, $\Theta^2, \ldots$ за графа от задача 3.4 и с тяхна помощ да се определи цикълът с отрицателна дължина.

Решение:

$$\Theta^0 = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}, \quad \Theta^1 = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 1 & 3 \end{pmatrix}, \quad \Theta^3 = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 2 & 2 \\ 3 & 1 & 2 \end{pmatrix}.$$

От задача 3.4 се вижда, че елементът $d_{33}^2 = -1$ (вж. решението), следователно в графа има цикъл с отрицателна дължина, равна на (-1), съдържащ върха x_3. Елементът $\theta_{33}^3 = 2$, т.е. предхождащ за връх 3, в този цикъл е връх 2. Елементът $\theta_{32}^3 = 1$, т.е. следващият предхождащ е връх 1. Елементът $\theta_{31}^3 = 3$ — край, стигнахме отново във връх 3. И така, цикълът $(3,1)$, $(1,2)$, $(2,3)$ има тегло (-1).

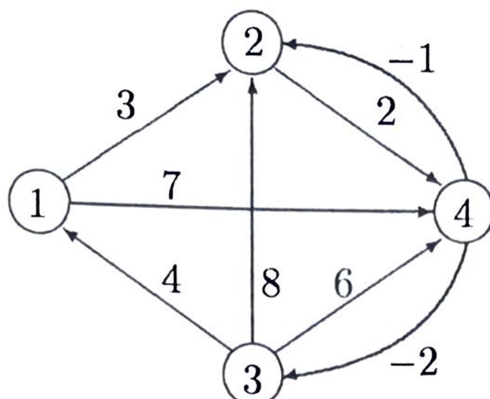
Една проста модификация на предложения метод е следната. Когато разстоянието между i-тия и j-тия се подобрява при преминаване през m-тия връх, в Θ^m елементът θ_{ij}^m става равен на m. В противен случай стойността на θ_{ij}^m остава същата. В този случай вместо (3.6), всъщност се ползват формулите

$$\theta_{ij}^m = \begin{cases} \theta_{ij}^{m-1}, & \text{при } d_{ij}^{m-1} \le d_{im}^{m-1} + d_{mj}^{m-1}, \\ m, & \text{при } d_{ij}^{m-1} > d_{im}^{m-1} + d_{mj}^{m-1}. \end{cases}$$

В случая разликата в анализа и "прочита" на последната Θ^n матрица се състои в следното: Елементът θ_{ij}^n дава връх от $(i-j)$ пътя. За разлика от преди, не е задължително този връх да се явява предпоследен за $(i-j)$ пътя.

Ще илюстрираме тази модификация като разгледаме следния граф G:

Граф G:



$$D^0 = \begin{pmatrix} 0 & 3 & \infty & 7 \\ \infty & 0 & \infty & 2 \\ 4 & 8 & 0 & 6 \\ \infty & -1 & -2 & 0 \end{pmatrix}; \qquad \Theta^0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix};$$

$$D^{\boxed{1}} = \begin{pmatrix} 0 & 3 & \infty & 7 \\ \infty & 0 & \infty & 2 \\ 4 & \boxed{7} & 0 & 6 \\ \infty & -1 & -2 & 0 \end{pmatrix}; \qquad \Theta^1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & \boxed{1} & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix};$$

$$D^{\boxed{2}} = \begin{pmatrix} 0 & 3 & \infty & \boxed{5} \\ \infty & 0 & \infty & 2 \\ 4 & 7 & 0 & 6 \\ \infty & -1 & -2 & 0 \end{pmatrix}; \qquad \Theta^2 = \begin{pmatrix} 1 & 1 & 1 & \boxed{2} \\ 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix};$$

$$D^{\boxed{3}} = \begin{pmatrix} 0 & 3 & \infty & 5 \\ \infty & 0 & \infty & 2 \\ 4 & 7 & 0 & 6 \\ \boxed{2} & -1 & -2 & 0 \end{pmatrix}; \qquad \Theta^3 = \begin{pmatrix} 1 & 1 & 1 & 2 \\ 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ \boxed{3} & 4 & 4 & 4 \end{pmatrix};$$

$$D^{\boxed{4}} = \begin{pmatrix} 0 & 3 & \boxed{3} & 5 \\ \boxed{4} & 0 & \boxed{0} & 2 \\ 4 & \boxed{5} & 0 & 6 \\ 2 & -1 & -2 & 0 \end{pmatrix}; \qquad \Theta^4 = \begin{pmatrix} 1 & 1 & \boxed{4} & 2 \\ \boxed{4} & 2 & \boxed{4} & 2 \\ 3 & \boxed{4} & 3 & 3 \\ 3 & 4 & 4 & 4 \end{pmatrix}.$$

Разстоянието от 2-ри до 1-ви връх е с дължина $d^4_{21} = 4$. По-

ради $\theta_{21}^4 = 4$, следва че в (2 – 1) **НКП** се минава и през връх номер 4. Тъй като $\theta_{24}^4 = 2$, следва че от връх 2 до връх 4 **НКП** е директен. Но тъй като $\theta_{41}^4 = 3$, следва че от връх 4 до връх 1 се минава през връх 3. Тъй като $\theta_{43}^4 = 4$ и $\theta_{31}^4 = 3$, следва че **НКП** е 2431.

Освен тази техника за определяне на **НКП**, предложена от Ху [19], съществува още един *външен (terminal) метод*, наречен така, тъй като се прилага след завършване работата на алгоритъма на Флойд. При този метод матрицата D^n е вече получена и числата (елементите) θ_{ij} се определят с помощта само на матриците D^0 и D^n. За целта е достатъчно, ако $d_{it}^n + d_{tj}^0 = d_{ij}^n$ за някое t, да се положи $\theta_{ij} = t$.

Алгоритъм на Данциг за намиране на всички НКП

Ще изложим още един, твърде сходен алгоритъм за търсене на всички най-кратки пътища. При този алгоритъм по същество се извършват същите операции, както в алгоритъма на Флойд, но в друг ред. В този случай матрицата D^m, $m \geq 1$ е с размери $m \times m$ (а не $n \times n$, както беше при Флойд), като от матрицата D^0 се определя матрицата D^1, от нея D^2 и т.н. от D^{n-1} се определя матрицата D^n.

При пресмятане на матриците D^m, $m \geq 1$ се използват на практика елементите на предишната матрица D^{m-1} и елементите на матрицата D^0.

ИДЕЯ НА АЛГОРИТЪМА (НА ДАНЦИГ). Всяка следваща матрица D^m съдържа един ред и стълб в повече от предишната матрица D^{m-1} $(m > 1)$. Елементите на D^m, които не са от последния ред и стълб се пресмятат точно както в алгоритъма на Флойд. Елементите d_{ij}^m при $i = m$ или $j = m$ се пресмятат, като се използват следните съображения. Най-краткият път от връх i до връх m с използване на първите m върха като междинни не може да съдържа като междинен върха m (липсват цикли с отрицателна дължина). Поради това, **НКП** $(i - m)$ се състои от две части. Първата част е **НКП** $(i - j)$, където $j < m$ и се използват за междинни първите $m-1$ върха, а втората част — най-късата дъга от върха j до върха m.

Аналогично най-краткият път от върха m до върха i с използване на първите m върха за междинни има също две части. Първата — това е най-късата дъга (m, j), $j < m$, а втората част — **НКП** $(j - i)$, в които за междинни се използват първите $m-1$ върха. Елементите d_{mm}^m се полагат равни на нула.

ОПИСАНИЕ НА АЛГОРИТЪМА НА ДАНЦИГ. *СТЪПКА 1.* Всички върхове на графа се номерират с естествените числа от 1 до n. Определя се матрицата $D^0_{n\times n}$, като елементите ѝ d^0_{ij} са дължините на най-късата дъга от връх i до връх j. При отсъствие на дъга (i,j) се полага $d^0_{ij} = \infty$.

СТЪПКА 2. За всяко $m = 1, 2, ..., n$ се определят съответните матрици D^m, чрез D^{m-1} и D^0 по следния начин.

$$d^m_{ii} = 0 \text{ за всяко } i \text{ и всяко } m. \tag{3.7}$$

$$d^m_{ij} = \min\{d^{m-1}_{ij},\ d^{m-1}_{im} + d^{m-1}_{mj}\}, \text{ при } i,j = 1,\ 2, ...,\ m-1. \tag{3.8}$$

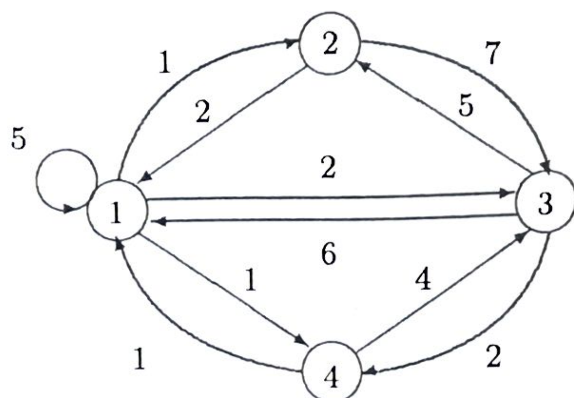
$$d^m_{im} = \min_{j=1,2,\dots,m-1}\{d^{m-1}_{ij} + d^0_{jm}\}, \text{ при } i = 1,\ 2, ...,\ m-1. \tag{3.9}$$

$$d^m_{mj} = \min_{i=1,2,\dots,m-1}\{d^0_{mi} + d^{m-1}_{ij}\}, \text{ при } j = 1,\ 2, ...,\ m-1. \tag{3.10}$$

ЗАДАЧА 3.6. Да се докаже, че алгоритъмът на Данциг и алгоритъмът на Флойд извършват еднакъв брой операции.

Решение: Уравнението (3.8) се използва и в двата алгоритъма. Уравненията (3.9) и (3.10) от алгоритъма на Данциг $m-1$ пъти повтарят (3.5) от алгоритъма на Флойд.

ЗАДАЧА 3.7. [1] Да се определят с алгоритмите на Флойд и Данциг най-кратките пътища между всеки два върха в следния граф.



$$D^0 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{pmatrix};$$

Решение: Алгоритъм на Флойд. Елементите на матрица D^1 се определят по следния начин.

$d^1_{ij} = \min\{d^0_{ij},\ d^0_{i1} + d^0_{1j}\}$	Път
$d^1_{11} = d^0_{11} = 0$	
$d^1_{12} = d^0_{12} = 1$	$(1,2)$
$d^1_{13} = d^0_{13} = 2$	$(1,3)$
$d^1_{14} = d^0_{14} = 1$	$(1,4)$
$d^1_{21} = d^0_{21} = 2$	$(2,1)$
$d^1_{22} = 0$	
$d^1_{23} = \min\{d^0_{23},\ d^0_{21} + d^0_{13}\} = \min\{7,\ 2+2\} = 4$	$(2,1),\ (1,3)$
$d^1_{24} = \min\{d^0_{24},\ d^0_{21} + d^0_{14}\} = \min\{\infty,\ 2+1\} = 3$	$(2,1),\ (1,4)$
$d^1_{31} = d^0_{31} = 6$	$(3,1)$
$d^1_{32} = \min\{d^0_{32},\ d^0_{31} + d^0_{12}\} = \min\{5,\ 6+1\} = 5$	$(3,2)$
$d^1_{33} = 0$	
$d^1_{34} = \min\{d^0_{34},\ d^0_{31} + d^0_{14}\} = \min\{2,\ 6+1\} = 2$	$(3,4)$
$d^1_{41} = d^0_{41} = 1$	$(4,1)$
$d^1_{42} = \min\{d^0_{42},\ d^0_{41} + d^0_{12}\} = \min\{\infty,\ 1+1\} = 2$	$(4,1),\ (1,2)$
$d^1_{43} = \min\{d^0_{43},\ d^0_{41} + d^0_{13}\} = \min\{4,\ 1+2\} = 3$	$(4,1),\ (1,3)$
$d^1_{44} = 0$	

$$D^1 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix} ; \quad D^2 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix} ;$$

$$D^3 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 6 & 5 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix} ; \quad D^4 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix} .$$

Най-кратките пътища, съответно са:

$$\begin{bmatrix} 0; & (1,2); & (1,3); & (1,4) \\ (2,1); & 0; & (2,1),(1,3); & (2,1),(1,4) \\ (3,4),(4,1); & (3,4),(4,1),(1,2); & 0; & (3,4) \\ (4,1); & (4,1),(1,2); & (4,1),(1,3); & 0 \end{bmatrix} .$$

Алгоритъм на Данциг:

$$D^0 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 7 & \infty \\ 6 & 5 & 0 & 2 \\ 1 & \infty & 4 & 0 \end{pmatrix} ; \ D^1 = (d^1_{11}) = (0).$$

Елементите на матрицата D^2 пресмятаме с помощта на (3.7) — (3.10).

Елементи на D^2	Път
$d^2_{11} = 0$	
$d^2_{12} = \min\{d^1_{11} + d^0_{12}\} = 0 + 1 = 1$	$(1,2)$
$d^2_{22} = 0$	
$d^2_{21} = \min\{d^0_{21} + d^1_{11}\} = 2 + 0 = 2$	$(2,1)$

Следователно за матрицата D^2 получихме

$D^2 = \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}$. Сравнете получената матрица със съответната част на матрицата, получена по метода на Флойд.

Пресмятаме елеменитe на матрицата D^3 и получаваме

Елементи на D^3	Път
$d^3_{13} = \min\{d^2_{11} + d^0_{13},\ d^2_{12} + d^0_{23}\} = \min\{0+2,\ 1+7\} = 2$	$(1,3)$
$d^3_{23} = \min\{d^2_{21} + d^0_{13},\ d^2_{22} + d^0_{23}\} = \min\{2+2,\ 0+7\} = 4$	$(2,1),\ (1,3)$
$d^3_{31} = \min\{d^0_{31} + d^2_{11},\ d^0_{32} + d^2_{21}\} = \min\{6+0,\ 5+2\} = 6$	$(3,1)$
$d^3_{32} = \min\{d^0_{32} + d^0_{22},\ d^0_{31} + d^2_{12}\} = \min\{5+0,\ 6+1\} = 5$	$(3,2)$
$d^3_{12} = \min\{d^2_{12},\ d^3_{13} + d^3_{32}\} = \min\{1,\ 2+5\} = 1$	$(1,2)$
$d^3_{21} = \min\{d^2_{21},\ d^3_{23} + d^3_{31}\} = \min\{2,\ 4+6\} = 2$	$(2,1)$
$d^3_{11} = d^3_{22} = d^3_{33} = 0$	

По този начин получаваме

$$D^3 = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 4 \\ 6 & 5 & 0 \end{pmatrix} \quad \text{и} \quad D^4 = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 0 & 4 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 2 & 3 & 0 \end{pmatrix}.$$

Най-кратките пътища при алгоритъма на Данциг, т.е. съвкупността от дъги на най-краткия път, се определят по външния метод, който, както отбелязахме при алгоритъма на Флойд, ползва само D^0 и D^n. Вътрешният метод също може да се използва, но неговото реализиране при този алгоритъм е малко по различно, отколкото при Флойд. За подробности вж. [1].

4. Алгоритъм за търсене на K-ти по дължина пътища между два върха s и t

За краткост първия, втория и т.н. k-тия по дължина път ще наричаме съответно *първи, втори, и т.н. k-ти* **НКП**. В много оптимизационни задачи се търси **НКП** с допълнителни свойства. Такава задача може да се разглежда като задача за **НКП** с допълнителни ограничения или като многоцелева задача. В тези случаи обаче се стига до редица усложнения и много често обемът на изчисления е твърде голям. Когато допълнителните свойства са субективни или нестрого формулирани, се оказва много по-просто и ефективно да се намерят k-тите **НКП** и от тях да се избере онзи, който притежава търсените свойства. Стига разбира се да разполагаме с ефективен метод за намиране на k-ти $(s - t)$ **НКП**.

Ще предполагаме, че търсим само прости пътища, т.е. пътища, в които няма повтарящи се върхове на графа (в повечето практически задачи се търсят прости пътища). Без това изискване задачата става доста по-проста - вж. [20], [21] и др. Обръщаме внимание на факта, че **НКП** очевидно е прост път, когато в графа няма цикли с отрицателни тегла, но 2-ят, 3-ят и т.н. k-ят **НКП** не е задължително да бъде прост път (дори всички дъги да са с положителни тегла).

Ще опишем *алгоритъма на Йен [22]* за намиране на k-те прости **НКП** (вж. и [2]).

ИДЕЯ НА АЛГОРИТЪМА. Нека $P^k = s,\ x_2^k,\ x_3^k, ...,\ x_{q_k}^k,\ t$ е k-тия по дължина път от s до t. Да разгледаме пътя P_i^k, наречен "*отклонение от пътя P^{k-1} във върха i*", определен по следния начин: P_i^k е **НКП**, съвпадащ с P^{k-1} от s-тия до i-тия връх, а след това отиващ във връх, различен от $(i+1)$-я връх на вече построените **НКП** P^j, $j = 1,\ 2, ...,\ k-1$, които имат същите начални подпътища $(s - i)$, както и P^{k-1}. Тъй като P_i^k достига до върха t по най-късия подпът, несъдържащ върховете $s,\ x_2^{k-1},\ x_3^{k-1}, ...,\ x_i^{k-1}$ (участващи във формирането на първата част на пътя P_i^k), то следва, че пътят P_i^k е прост път.

Първата част на пътя P_i^k, т.е. подпътят $s,\ x_2^k,\ x_3^k, ...,\ x_i^k$ (съвпадащ със $s,\ x_2^{k-1},\ x_3^{k-1}, ...,\ x_i^{k-1}$) се нарича *корен на отклонението* P_i^k, а вторият подпът $x_i^k, ...,\ t$ се нарича *разклонение на* P_i^k. Коренът на P_i^k се бележи с R_i^k, а разклонението — със S_i^k.

Алгоритъмът намира най-напред P^1, т.е. **НКП**, като използва някои от разглежданите вече алгоритми. Този път и всички останали k-ти **НКП** се съхраняват в един списък L_0. Пътят P^k най-общо се намира след намирането на P^1, $P^2, ..., P^{k-1}$.

ОПИСАНИЕ НА АЛГОРИТЪМА НА ЙЕН [2]. *СТЪПКА 1.* Намерете P^1. Положете $k = 2$. Ако пътят P^1 е единствен, включете го в списъка L_0 и преминете към *СТЪПКА 2.* Ако съществуват няколко такива пътища, но по-малко от K, включете в списъка L_0 един от тях, а останалите в списъка L_1 и преминете към *стъпка 2.* Ако броят на **НКП** P^1 е поне K, спрете — задачата е решена, край.

СТЪПКА 2. Намерете всички отклонения P_i^k на $(k-1)$-я **НКП** P^{k-1} при $i = 1, 2, ..., q_{k-1}$, изпълнявайки за всяко i *стъпки* 3 — 6.

СТЪПКА 3. Проверете съвпада ли подпътят, образуван от първите i върха на P^{k-1}, с подпътя, образуван от първите i върха на всеки от пътищата P^j, $j = 1, 2, ..., k-1$. Ако е така за теглото на дъгата (x_i^{k-1}, x_{i+1}^j) положете $d(x_i^{k-1}, x_{i+1}^j) = \infty$. В противен случай не променяйте нищо и преминете към *стъпка 4.*

СТЪПКА 4. С алгоритъма за намиране на **НКП** намерете най-краткия път S_i^k от x_i^{k-1} до t, като изключите от разглеждане върховете s, x_2^{k-1}, $x_3^{k-1}, ..., x_i^{k-1}$. Ако тези пътища са няколко — вземете един от тях.

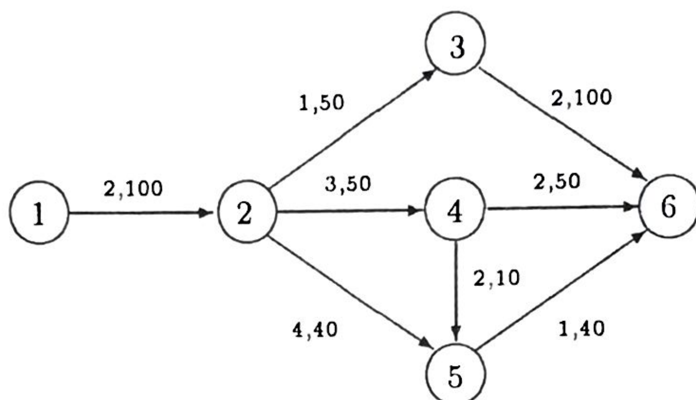
СТЪПКА 5. Постройте P_i^k, т.е. съединете $R_i^k = (s, x_2^{k-1}, x_3^{k-1}, ..., x_i^{k-1})$ със S_i^k и съхранете P_i^k в списъка L_1.

СТЪПКА 6. Заменете теглата на дъгите, които са изменени в *стъпка 3* с първоначалните им значения и се върнете на *стъпка 3.*

СТЪПКА 7. Намерете най-кратикя път в списъка L_1 и го обозначете с P^k. Преместете го в списъка L_0. Ако $k = K$, край на алгоритъма. В L_0 се съдържат търсените K-ти **НКП**. В противен случай, т.е. ако $k < K$, положете $k = k+1$ и се върнете на *стъпка 2.* Ако в L_1 има h на брой **НКП** (т.е. повече от един), то поместете в L_0 всеки от тях и продължете, както е указано по-горе, докато броят на пътищата в L_0 не стане поне K.

ЗАДАЧА 3.8. Командирован трябва да пътува със самолет от град 1 до град 6 за възможно най-малко часове (по здравословни причини например),

като полагащите му се в заповедта командировъчни са в рамките на $200. Възможните маршрути са илюстрирани със следния граф.



Първите тегла на всяка от дъгите са продължителността на полета, а вторите тегла — цената на полета. Да се намери **НКП**, който трябва да избере командированият, без да надхвърля лимита за пътни разноски.

Отг.: Пътят е: $(1,2)$, $(2,4),(4,6)$ — 7 часа, $200 или
$(1,2)$, $(2,5),(5,6)$ — 7 часа, $180.

КОМЕНТАР И ОБОСНОВКА НА АЛГОРИТЪМА (НА ЙЕН).

Както вече споменахме, пътищата, които алгоритъмът строи, са прости. Пътят P^k трябва да бъде отклонение на i-тия етап $(i \geq 1)$ от един от пътищата P^1, $P^2, \ldots$, P^{k-1}. Ето защо е необходимо да се построят най-кратките отклонения от всеки от горните пътища P^j и от тях да се избере най-краткото отклонение. То всъщност ще бъде пътят P^k. Това точно прави и даденият алгоритъм.

Обърнете внимание, че при k-тата итерация всички най-кратки отклонения от P^j, $j = 1, 2, \ldots, k-2$, вече са включени в списъка L_1 и е достатъчно да се намери само едно отклонение от P^{k-1}, което да се включи в наличния списък.

Смяната на теглата в *СТЪПКА 3* се прави с цел да не се получи отново P^j като отклонение в точка i от пътя P^{k-1}. Това е възможно, тъй като при $j < k-1$ теглото на пътя P^j не надхвърля теглото на пътя P^{k-1}. Смяната на теглата гарантира отсъствието на тази възможност.

Изпълнявайки *СТЪПКА 5* на алгоритъма, всеки породен път P_i^k се включва в списъка L_1. Тогава очевидно на k-тата итерация този списък ще съдържа не повече от $K-k+1$ най-кратки отклонения P_i^k.

Изложените до тук методи за търсене на **НКП** се отнасяха за произволни графи. Много често в практиката се налага да

се търсят **НКП** за ориентирани и ациклични графи, при които горните методи разбира се са също приложими, но се явяват твърде неикономични и тромави. За ориентираните ациклични графи съществува значително по-ефективен алгоритъм за намиране на **НКП**. По-късно ще разгледаме един такъв алгоритъм (вж. параграф 6).

5. Задачи, свеждащи се до търсене на НКП — път с най-голяма пропускателна способност и най-надежден път

Път с най-голяма пропускателна способност

ПРИМЕР 3.5. Да разгледаме мрежа от мостове между различни пунктове. Известна е пропускателната способност на мостовете, свързващи всеки два пункта. Да се определи максималното количество, което може да бъде пропуснато между пунктовете s и t в разглежданата мрежа (еднократно).

Очевидно могат да бъдат формулирани редица задачи, аналози на разглеждания пример.

Ако на върховете на графа G съпоставим дадените пунктове, а дъгите и техните тегла са съответно мостовете и тяхната пропускателна способност, то очевидно проблемът се свежда до следното: *Да се намери такъв път между върховете s и t, в който дължината на най-късата дъга е максимална.* В досегашните задачи за търсене на **НКП**, дължината на пътя се определяше като сума от дължините на участващите в него дъги. В алгоритмите за намиране на **НКП** беше използвана триместната операция (3.5), чието приложение n-пъти към матрицата на теглата даваше матрицата, определяща дължините на **НКП**. Тази операция се явява частен случай на една по-обща триместна операция:

$$d_{ij}^{m} = OP[d_{ij}^{m-1},\ d_{im}^{m-1} \otimes d_{mj}^{m-1}], \tag{3.11}$$

където d е функция, подлежаща на оптимизация, а $\otimes$ е някаква обща операция. Очевидно задачата за намиране на **НКП** и задачата, формулирана в пример 3.5, са твърде сходни и за решаването им може да се използват едни и същи алгоритми. Различието ще се състои само в това, че операциите събиране и търсене на min в 3.5 трябва да се заменят съответно с операциите търсене на min и търсене на max. Освен това, ако в алгоритмите на Флойд и Данциг заменим само операцията събиране

с операцията търсене на min, очевидно съответните модификации на алгоритмите ще генерират път с *минимална дължина на най-късата дъга (път с минимална пропускателна способност)*.

Много често задачата от пример 3.5 се нарича задача за "тесните места". Прилагането на алгоритмите на Флойд и Данциг за решаване на задачата за тесните места се извършва след като всички елементи на матрицата D^0, съответни на липсващи в графа дъги се положат равни на $-\infty$ (диагоналните също).

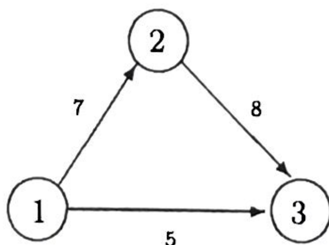
ЗАДАЧА 3.9. Да се докаже, че пропускателната способност на пътя с най-голяма пропускателна способност от s до t е равна на

$$\min_K [\max_{(x_i,x_j)\in K}(d_{ij})],$$

където K е $(s-t)$-разрез в множеството от дъги.

По-късно, когато разглеждаме потокови алгоритми (алгоритъм на Форд -Фалкерсон за максимален поток), ще видим, че тази задача се явява минимаксен вариант на теоремата за максимален поток и минимален разрез.

ЗАДАЧА 3.10. Да се определят пропускателните способности на пътищата в мрежата



Решение:

$$D^0 = \begin{pmatrix} -\infty & 7 & 5 \\ -\infty & -\infty & 8 \\ -\infty & -\infty & -\infty \end{pmatrix}, \quad D^1 = \begin{pmatrix} -\infty & 7 & 5 \\ -\infty & -\infty & 8 \\ -\infty & -\infty & -\infty \end{pmatrix},$$

$$D^2 = \begin{pmatrix} -\infty & 7 & 7 \\ -\infty & -\infty & 8 \\ -\infty & -\infty & -\infty \end{pmatrix}, \quad D^3 = \begin{pmatrix} -\infty & 7 & 7 \\ -\infty & -\infty & 8 \\ -\infty & -\infty & -\infty \end{pmatrix}.$$

Най-надежден път

Да разгледаме сега задача, при която теглото α_{ij} на дъгата (x_i, x_j) представлява нейната надежност, т.е. вероятността дъгата да съществува в графа или в случаите на физичеси системи

— вероятността тя да се намира в работоспособно състояние. Под надеждност на пътя P от s до t, се разбира

$$\alpha(P) = \prod_{(x_i,x_j)\in P} \alpha_{ij}. \tag{3.12}$$

Проблемът да се намери най-надеждният път от s до t, може да се сведе към задачата за търсене на $(s-t)$ **НКП**, вземайки в качеството на тегло d_{ij} за дъгата (x_i, x_j) величината $d_{ij} = -\log\alpha_{ij}$. Ако логаритмуваме двете страни на (3.12), ще получим

$$\lg\alpha(P) = \sum_{(x_i,x_j)\in P} \lg\alpha_{ij} = -\sum_{(x_i,x_j)\in P} d_{ij}.$$

Оттук се вижда, че $(s-t)$ **НКП** с матрица на теглата (d_{ij}) ще бъде в същото време и най-надеждният път с матрица (α_{ij}), а надеждността на пътя ще бъде антилогаритъмът от неговата дължина.

По-общо, ако теглото на всяка дъга е някакво реално число, наречено *коефициент на усилване на дъгата*, и под *коефициент на усилване на пътя* се разбира произведението от коефициентите на усилване на неговите дъги, е ясно, че с непосредствено използване на триместната операция

$$\alpha_{ij}^{m} = \max\{\alpha_{ij}^{m-1},\ \alpha_{im}^{m-1}.\alpha_{mj}^{m-1}\} \tag{3.13}$$

можем да търсим път с максимален коефициент на усилване, използвайки алгоритмите на Флойд и Данциг. За целта е достатъчно дължините на дъгите на графа да считаме за коефициенти на усилване в съответната дъга и в алгоритмите за **НКП**, операциите събиране и търсене на минимум да заменим съответно с операциите умножение и търсене на максимум. Както задачата за търсене на **НКП** не може да бъде решена при наличие на цикъл с отрицателна дължина в изходния граф, така и задачата за намиране на път с максимален коефициент на усилване не може да бъде решена, ако съществува цикъл с коефициент на усилване по-голям от 1. (Минавайки през такъв цикъл непрекъснато, ще се формират непрости пътища с неограничен брой пъти, ще се формират непрости пътища с неограничено голям коефициент на усилване.)

Ясно е, че ако търсим път с *минимален коефициент на усилване*, е достатъчно в (3.13) операцията търсене на максимум да

се замени с търсене на минимум. Аналогично задачата за търсене на път с минимален коефициент на усилване е нерешима, ако в изходния граф съществува цикъл с коефициент на усилване, по-малък от -1. (Наличието на такъв цикъл позволява неограничен брой пъти да се минава по този цикъл и така да се получават непрости пътища с произволно малък коефициент на усилване.)

Най-общо алгоритмите на Флойд и Данциг могат да бъдат модифицирани, като използваните в тях операции събиране и търсене на минимум се заменят и с други операции, при съответната интерпретация на дължините на дъгите на изходния граф, като по този начин се намират оптимални пътища и в друг смисъл (вж. [1] и [2]).

2.4. Сложност на алгоритми и задачи

Да разгледаме оптимизационната задача

$$L(X) \to \max\ (\min), \quad \text{където} \quad X \in D. \tag{4.1}$$

Тъй като областта D, в която се търси екстремумът на целевата функция $L(X)$, е най-често множество с краен брой елементи, задачата по принцип е решима. Пълното изброяване на елементите на D гарантира намирането на решение. Това твърде често не само е неефективно, но и прави невъзможно получаването на решение в реално време.

ПРИМЕР 4.1. (*Задача за търговския пътник*) Съществуват n на брой града и са известни разстоянията между всеки два от тях ($d_{ij} \in \mathbb{Z}^+$ е разстоянието между i-тия и j-тия град). Търговец трябва да намери затворен маршрут, преминавайки през всеки град точно по веднъж така, че да измине минимално разстояние.

Всички циклични пермутации π, състоящи се от n обекта, ще представляват маршрут, ако интерпретираме $\pi(j)$ като град, посещаван след града j, $j = 1, 2, \ldots, n$, като за дължината на π ще имаме

$$\sum_{j=1}^{n} d_{j\pi(j)}.$$

В този случай методът на пълното изчерпване е приложим — намират се всички маршрути, изчислява се тяхната дължина

и се избира най-краткият. Броят на тези маршрути (обхождания), преминаващи по веднъж през всеки от тези n града е, $\frac{(n-1)!}{2}$.

Решаването на една конкретна задача от този тип изисква около $n!$ елементарни команди (стъпки). Дори при $n \approx 50$ ще бъдат необходими милиони години за получаване на това "решение" на задачата. Следователно целта е да се намерят ефективни (бързи) алгоритми. Въпросът е и: "Дали за всяка оптимизационна задача съществуват такива алгоритми?"

В този параграф съвсем накратко без претенции за изчерпателност ще разгледаме някои въпроси, свързани със сложността на алгоримтите и задачите, които те решават.

Да означим с P (*problem*) някаква оптимизационна задача, от типа (4.1). Всяка такава задача се характеризира с някакви параметри — коефициентите на целевата функция $L(X)$ и коефициентите на ограниченията, определящи D. Задачата има и размери, които са някаква функция d на параметрите ѝ. При оптимизационни задачи P върху граф $G = (V, A)$ размерите са обикновено $d(|V|)$ или пък $d(|V|, |A|)$. При задачата на линейното оптимиране очевидно $d = d(m, n)$, където m е броят на ограниченията, n е броят на променливите.

Да разгледаме следната задача:

$$P : L(x_1, x_2, x_3) = c_1x_1 + c_2x_2 + c_3x_3 \to \min\ (\max)$$

$$\begin{aligned} &a_{11}.x_1 + a_{12}.x_2 + a_{13}.x_3 \le b_1 \\ &a_{21}.x_1 + a_{22}.x_2 + a_{23}.x_3 \le b_2 \\ &a_{31}.x_1 + a_{32}.x_2 + a_{33}.x_3 \le b_3 \\ &x_1,\ x_2,\ x_3 \ge 0. \end{aligned} \tag{4.2}$$

Ако зададем конкретни стойности на коефициентите в P, ще получим една *конкретна (индивидуална) задача* p.

Да означим с $P(d)$ *множеството от всички индивидуални задачи от тип P с размери d*. Ако $p \in P(d)$ и ALG е алгоритъм на решаване на p, под *сложност на ALG за решаване на p* ще разбираме броя операции (събиране, умножаване, присвояване, сравняване и др.), необходими за пълното решаване на задачата p чрез ALG. Сложността на алгоритъма ще бележим с

$$complexity\ (ALG,\ p). \tag{4.3}$$

Ако ALG е алгоритъм, който решава всички индивидуални задачи от типа P, под *сложност на ALG за решаване на P* с размери d, ще разбираме числото

$$(4.4) \qquad complexity\,(ALG,\ P) = \max_{p \in P(d)}\ complexity\,(ALG, p).$$

Ако с $ALG(P)$ обозначим множеството на всички известни алгоритми за решаване на P с размери d, *под сложност на P (сложност на задача)* разбираме числото

$$(4.5) \qquad complexity\,(P) = \min_{ALG \in ALG(P)}\ complexity\,(ALG, P).$$

Сложността на един алгоритъм ALG за решаване на конкретна задача p може да се определи чрез изброяване на извършваните операции. Сложността на ALG за решаване на P е по-трудоемък процес, който използва (4.4), и този процес зависи от задачата P и алгоритъма ALG. При задачата (4.2) стойностите на (4.3) и (4.4) не се различават съществено.

От казаното следва, че сложността на един алгоритъм е число, еднозначно свързано с него. Не така стоят нещата, когато определяме сложността на P (сложност на задача). Тази оценка има субективен характер, тъй като не е ясно дали познаваме всички алгоритми, решаващи проблема P. Дори да предположим, че познаваме всички алгоритми, няма гаранция, че няма да бъде открит нов алгоритъм, който да промени оценката (стойността на минимума в (4.5)).

Задачите, както и алгоритмите за тяхното решаване, обикновено се сравняват помежду си чрез понятието *сложност*. Тъй като при задачи с малки размери пълното изброяване на всички варианти е практически възможно, то интерес представлява асимптотичното поведение на сложността, когато размерите d на задачата P растат неограничено. В случаите на голяма размерност реално приложими са алгоритмите с най-малка сложност.

При изследване скоростта на нарастване на сложността като функция на размерите на задачата, се използва следният *символ на Ландау*: Нека $f(n)$ и $g(n)$ са функции, дефинирани в множеството на естествените числа, получаващи положителни стойности и

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = +\infty.$$

Означаваме $f(n) = O(g(n))$ точно тогава, когато съществува константа $c > 0$, така че $f(n) \leq c.g(n)$, за всяко n. Символът на

Ландау има следните свойства:

а) $O(f(n)) + O(f(n)) = O(f(n))$;
б) $O(f(n)).O(g(n)) = O(f(n).g(n))$;
в) $O(O(f(n))) = O(f(n))$;
г) $O(c.f(n)) = O(f(n)),\ c > 0$;
д) $O(n^p + n^q) = O(n^p)$, при) $p > q > 0$;
е) $O(n^p) + O(n^q) = O(n^p)$, при) $p > q > 0$;
ж) $O(1) \le c,\ c > 0$.

Използва се и следното означение

$$f(n) = \Omega(g(n)),$$

ако съществува константа $c > 0$, така че $f(n) \geq c.g(n)$ за достатъчно големи n.

С други думи, при сравняването на скоростта на нарастване на две функции $f(n)$ и $g(n)$, получаващи неотрицателни стойности, са удобни следните означения:

$$f(n) = O(g(n)) \iff \exists c,\ n_0 > 0, \text{ такива че за } \forall n \geq n_0 \quad f(n) \leq c.g(n)$$
$$f(n) = \Omega(g(n)) \iff \exists c,\ n_0 > 0, \text{ такива че за } \forall n \geq n_0 \quad f(n) \geq c.g(n)$$

Очевидно $f(n) = \Omega(g(n))$ тогава и само тогава, когато $g(n) = O(f(n))$. Символите $O(g(n))$ и $\Omega(g(n))$ се четат съответно "порядък, не по-голям от $g(n)$" и "порядък, не по-малък от $g(n)$".

Ако сложността на един алгоритъм е $O(g(n))$, казваме, че алгоритъмът "изисква" време от порядък $g(n)$.

По аналогичен начин се дефинират символите $O(g(n_1, \ldots, n_k))$ и $\Omega(g(n_1, \ldots, n_k))$ за функции на повече променливи.

Определената с помощта на тези символи сложност, се нарича *времева сложност* за разлика от *сложността по памет*, която определя обема памет, използвана от алгоритъма, като функция на размерността на задачата.

Записът $f(n) = O(g(n))$ не трябва да се схваща като равенство. Например от $f(n) = O(g(n))$ и $h(n) = O(g(n))$ не следва $f(n) = h(n)$.

Освен това се пише

$$f(n) = \Theta(g(n)),$$

ако съществуват константи c и c', така че $c.g(n) \leq f(n) \leq c'.g(n)$ за достатъчно големи n.

Релацията ρ, дефинирана чрез

$$f(n) \;\; \rho \;\; g(n) \Leftrightarrow f(n) = \Theta(g(n)),$$

очевидно е релация на еквивалентност, т.е. е рефлексивна, симетрична и транзитивна. Като такава тя разбива множеството от функции на непресичащи се класове на еквивалентност. Класът, съдържащ $f(n)$, т.е. множеството от всички функции $g(n)$, такива че $f(n) = \Theta(g(n))$, се нарича *скорост на нарастване на* $f(n)$. С други думи функциите от класа на еквивалентност имат едно и също асимптотично поведение.

С помощта на понятието *скорост на нарастването* на сложността на алгоритъма, може да се направи оценка за изискуемото време. Пресмятайки броя елементарни стъпки (аритметични операции, сравнения и т.п), необходими за изпълнение на алгоритъма, предполагайки че всяка операция изисква единица време, можем, оценявайки сложността, да казваме например, "изискуемо време $O(n^5)$".

С други думи, под стъпки на алгоритъма се имат предвид командите за пренос на думи от памет във буфер и обратно, аритметичните операции събиране, изваждане, умножение и деление, условните преходи, операциите вход-изход и косвената адресация.

Ясно е, че при тази дефиниция за стъпка на алгоритъма, сложността ще зависи от вида на машинните команди. На практика обаче, по-важна е не точната сложност на алгоритъма, а неговата асимптотична сложност, т.е. да можем да определим приблизителната скорост на увеличаване стъпките на алгоритъма, когато размерността на задачата неограничено расте (презумпцията е, че обемът на паметта е неограничен и всяка клетка от паметта може да съдържа произволно голямо цяло число).

Нека задачата P с размери d има според (4.5) оценка на сложността $O(f(d))$. Ако $f(d)$ при големи стойности на d е ограничена от полином на d, се казва, че P е *задача с полиномна сложност.* Аналогично се дефинира и *алгоритъм с полиномна сложност.* Класът на всички задачи с полиномна сложност се бележи с $\mathcal{P}$.

Когато $P \notin \mathcal{P}$, се казва, че *задачата P е с експоненциална сложност.* Класът на задачите с експоненциална сложност се бележи с $\overline{\mathcal{P}}$.

По аналогия, ако в оценката $O(f(d))$ за сложността на един алгоритъм функцията $f(d)$ не е ограничена от полином при нарастване на d, се казва, че *алгоритъмът е с експоненциална сложност*. Такава е сложността на симплекс-метода например.

ЗАДАЧА 4.1. Каква е сложността на алгоритмите (задачите) със следните оценки.

а) $O(n^2)$ б) $O(2^n)$ в) $O(n.3^n)$
г) $O(n \lg n)$ д) $O(n\sqrt{n})$ е) $O(n^{100})$

Отг.: а) полиномна б) експоненциална в) експоненциална
г) полиномна д) полиномна е) полиномна

Да разгледаме следните таблици ([23] и [13]):

n / compl	10	20	30	40	50	60
n	$0,00001$ сек	$0,00002$ сек	$0,00003$ сек	$0,00004$ сек	$0,00005$ сек	$0,00006$ сек
n^2	0,0001 сек	0,0004 сек	0,0009 сек	0,0016 сек	0,0025 сек	0,0036 сек
n^3	0,001 сек	0,008 сек	0,027 сек	0,064 сек	0,125 сек	0,216 сек
n^5	0,1 сек	3,2 сек	24,3 сек	1,7 мин	5,2 мин	13,0 мин
2^n	0,001 сек	1,0 сек	17,9 мин	12,7 дни	35,7 год.	366 века
3^n	0,059 сек	58 мин	6,5 год.	3855 века	2×10^8 века	$1,3 \times 10^{13}$ века

Таблица 4.1

сложност	на съвременен компютър	на 100 пъти по-бърз компютър	на 1000 пъти по-бърз компютър
n	d_1	$100\ d_1$	$1000\ d_1$
n^2	d_2	$10\ d_2$	$31,6\ d_2$
n^3	d_3	$4,64\ d_3$	$10\ d_3$
n^5	d_4	$2,5\ d_4$	$3,98\ d_4$
2^n	d_5	$d_5 + 6,64$	$d_5 + 9,97$
3^n	d_6	$d_6 + 4,19$	$d_6 + 6,29$

Таблица 4.2

Времената, дадени в тези таблици, се отнасят за компютър със скорост 10^6 операции в секунда.

При малки размери, например $n = 10$ и $n = 20$ е възможно за по-сложни задачи да е необходимо по-малко време (вж. n^5 и 2^n, и 3^n при $n = 10$, както и n^5 и 2^n при $n = 20$).

Нарастването на размерите обаче при задачи с експоненциална сложност води до "неудържимо" нарастване на времето (табл. 4.1).

В таблица 4.2 е дадено как влияе мощността на компютъра върху размерите на задачите с различна сложност, решими за един час. Оказва се, че колкото задачата е с по-голяма сложност, толкова по-малко компютърното бързодействие влияе върху размерите на задача, която може да се реши за 1 час.

При задачи с полиномна сложност увеличението е "пъти" (мултипликативна константа), докато при задачи с експоненциална сложност увеличението е с адитивна константа (вж. последните два реда на таблица 4.2).

Ето защо алгоритмите (задачите) с полиномна сложност се смятат за "добри", докато тези с експоненциална сложност се считат за "лоши" (трудоемки).

Разбира се, една задача със сложност $O(n^{50})$ не е практически решима за големи стойности на n. Статистиката показва обаче, че много често за задачите от класа $\mathcal{P}$ се намират нови и ефективни алгоритми за тяхното решаване. Не така стоят нещата при задачите с експоненциална сложност от класа $\overline{\mathcal{P}}$. Освен това, практиката сочи, че ако е известен класът на сложност на една задача, това дава добри насоки какъв алгоритъм да се търси за нейното решаване (особено в областта на дискретното оптимиране).

Да се върнем отново на задача от типа (4.1), т.е.

$$P: \ L(X) \to \max\ (\min), \text{ където } X \in D.$$

Казва се, че имаме:

- *оптимизационен вариант на* P, когато целта е да се намери $X^* \in D$, за което $L(X)$ достига максимум (минимум);

- *изчислителен вариант на* P, когато се търси само стойността на този максимум (минимум);

- *разпознавателен вариант на* P, когато се търси само отговор на въпроса "дали съществува $\overline{X} \in D$, така че $L(\overline{X}) \leq L$, където L е цяло число".

Очевидно решаването на оптимизационния вариант води до решаване на останалите два варианта на задачата.

Решението на разпознавателния вариант на P е с отговор "да" или "не" и с известни уговорки е възможно от него да се получи решението на изчислителния вариант. От решенията на последните два варианта на P обаче, не може да се получи решението на оптимизационния вариант на P.

1. Някои класове на сложност — $\mathcal{NP}$, $\mathcal{NPC}$, $\mathcal{NPI}$ и $\mathcal{NPH}$

> Една задача за разпознаване P принадлежи на класа $\mathcal{NP}$ (Notdeterministic Polynominal), когато за всяка конкретна (индивидуална) задача с решение "да", проверката за верността на това се извършва след полиномно ограничен брой операции.

а) Задачата на линейното оптимиране (канонична форма), както отбелязахме в глава I, е:

$$P:\ L(X) = \sum_{j=1}^{n} c_j x_j \to \max\ (\min)$$

при ограничения

$$\sum_{j=1}^{n} a_{ij} x_j = b_i,\ i = 1,\ 2, ...,\ m,$$

$$x_j \geq 0,\ j = 1,\ 2, ...,\ n.$$

Да разгледаме нейния разпознавателен вариант, т.е. ако L е дадена константа, съществува ли X, така че да са изпълнени ограниченията $AX = B$ и $CX \leq L$ (векторен запис).

Да разгледаме произволна конкретна задача p с отговор (решение) "да". За да се провери верността на това, трябва да се пресметне CX (скаларно произведение) и да се провери дали $CX \leq L$ — необходими са $2n$ операции. Като вземем предвид, че трябва да се провери и принадлежността на X към дефиниционната област, т.е. $AX = B$, $X \geq 0$, ще получим окончателно

оценката $O(mn)$ действия. Следователно за задачата на линейното оптимиране P, разглеждана по-горе, имаме $P \in \mathcal{NP}$.

б) Да разгледаме следната задача P: Вярно ли е, че в пълния неоринетиран граф с n върха, с дадени дължини на ребрата, всички обходи на търговския пътник от пример 4.1 имат дължина по-голяма или равна на L, където L е дадено положително число?

Ако отговорът е "да", проверката за верността му налага да се образуват всички такива обходи — $(n-1)!/2$ на брой, т.е. необходими са експоненциален брой действия, следователно, за тази задача P имаме $P \notin \mathcal{NP}$.

Класът $\mathcal{NP}$ не е празен и очевидно всяка задача с полиномна сложност се съдържа в него, т.е. $\mathcal{P} \subset \mathcal{NP}$. За много от задачите в класа $\mathcal{NP}$ е съмнително обаче дали са с полиномна сложност. До сега не е доказано нито равенството $\mathcal{P} = \mathcal{NP}$, нито неравенството $\mathcal{P} \neq \mathcal{NP}$.

Предполага се, че $\mathcal{P} \neq \mathcal{NP}$, но това, както и противоположното твърдение, може да се окажат недоказуеми.

Да разгледаме сега две задачи P_1 и P_2 от типа

$$P: \begin{cases} L_i(X) \to \max(\min), \quad i = 1,\ 2, \\ X \in D_i \end{cases}$$

и означим с $P_i(d_i)$ множеството от всички индивидуални задачи от вида P_i с размери d_i. За всяка задача p от това множество, със $S_i(p)$ ще бележим множеството от допустимите ѝ решения и

$$S_i(P_i(d_i)) = \bigcup_{p \in P_i(d_i)} S_i(p).$$

Казваме, че задачата P_1 *полиномно се трансформира* в P_2, когато съществува функция $f: P_1(d_1) \to P_2(d_2)$, за която:
1. Пресмятането стойностите на f става след полиномен брой операции;
2. За всяка задача $p_1 \in P_1(d_1)$ и $p_2 = f(p_1)$ са изпълнени
а) ако $S_2(p_2) = \emptyset$, то и $S_1(p_1) = \emptyset$;
б) ако $S_2(p_2) \neq \emptyset$, то съществува функция $g: S_2(P_2(d_2)) \to S_1(P_1(d_1))$, стойностите на която се пресмятат след полиномно ограничен брой действия и освен това

$$X_1 = g(X_2) \text{ и } X_2 \in S_2(p_2) \Rightarrow X_1 \in S_1(p_1).$$

Това, че P_1 полиномно се трансформира в P_2, ще бележим с $P_1 \propto P_2$.

ПРИМЕР 4.2. Да разгледаме стандартната и каноничната задача на линейното оптимиране

$$P_1: \begin{cases} L(X) \to \max\ (\min) \\ AX \leq B,\ X \geq 0 \end{cases} \text{ и } P_2: \begin{cases} L(X) \to \max\ (\min) \\ AX = B,\ X \geq 0. \end{cases}$$

Очевидно $P_1 \propto P_2$, тъй като

$$P_1: \begin{cases} L(X) \to \max\ (\min) \\ AX \leq B,\ X \geq 0 \end{cases} \xrightarrow[\text{въвеждане на допълн. пром-ви}]{f} P_2: \begin{cases} L(X) \to \max\ (\min) \\ AX + Y = B,\ X,\ Y \geq 0 \end{cases}$$

и след решаване на P_2 имаме

$$\begin{cases} P_2 \text{ няма решение} \\ P_2 \text{ има решение } (X,Y) \in R^{n+m} \end{cases} \xrightarrow[\text{игнориране на допълн. пром-ви}]{g} \begin{cases} P_1 \text{ няма решение} \\ P_1 \text{ има решение } X \in R^n. \end{cases}$$

Ще изложим схематично доказателствата на следните три теореми [13]:

▹ *ТЕОРЕМА 4.1. Ако $P_1 \propto P_2$ и $P_2 \in \mathcal{P}$, то и $P_1 \in \mathcal{P}$.*

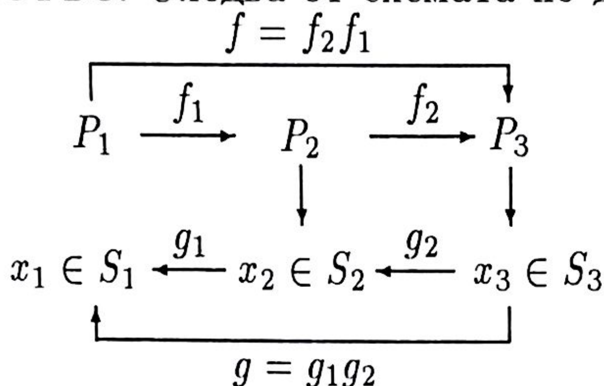
Доказателство: Следва от схемата

$$P_1 \xrightarrow[O(Q_1)]{f} P_2 \xrightarrow[\text{решима с оценка } O(Q_2)]{} X_2 \in P_2(d_2) \xrightarrow[O(Q_3)]{g} X_1 \in P_1(d_1),$$

където Q_i са полиноми. Следователно решението X_1 на P_1 се получава след $O(Q) = O(Q_1) + O(Q_2) + O(Q_3)$. ◃

▷ *ТЕОРЕМА 4.2. (транзитивност) Ако* $P_1 \propto P_2$ *и* $P_2 \propto P_3$, *то* $P_1 \propto P_3$.

Доказателство: Следва от схемата по-долу:



Да разгледаме сега класа на най-сложните задачи от $\mathcal{NP}$. Означават се с $\mathcal{NP}$-complete или $\mathcal{NPC}$. Всяка задача от този клас се нарича $\mathcal{NP}$-пълна (универсална) задача. Класът $\mathcal{NP}$-complete ($\mathcal{NPC}$) се дефинира като подклас на $\mathcal{NP}$ по следния начин:

$$\mathcal{NPC} = \{P \in \mathcal{NP} |\ \forall Q \in \mathcal{NP},\ Q \propto P\}.$$

◁

▷ *ТЕОРЕМА 4.3. Нека* P_1 *и* P_2 *са задачи от класа* $\mathcal{NP}$. *Тогава*

$$(P_1 \in \mathcal{NPC}) \wedge (P_1 \propto P_2) \Rightarrow P_2 \in \mathcal{NPC}.$$

Доказателство: Да вземем напълно произволна задача $Q \in \mathcal{NP}$. Поради $P_1 \in \mathcal{NPC}$, следва $Q \propto P_1$. От това, от $P_1 \propto P_2$ и теорема 4.2 следва $Q \propto P_2$.

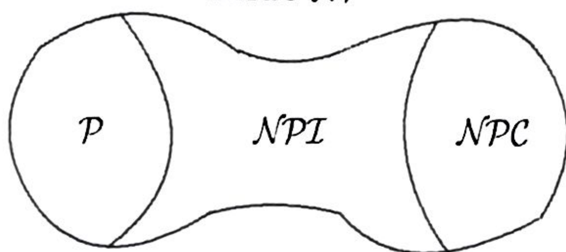
Като вземем предвид, че $P_2 \in \mathcal{NP}$ и че Q беше напълно произволна задача, за която получихме $Q \propto P_2$, следва верността на теоремата — $P_2 \in \mathcal{NP}$–complete. ◁

С използване на хипотезата $\mathcal{P} \neq \mathcal{NP}$ и теорема 4.1 се получава $P \cap \mathcal{NP}$ – complete $= \emptyset$. Освен това $\mathcal{NP} \setminus (\mathcal{P} \cup \mathcal{NPC}) \neq \emptyset$. Да означим с $\mathcal{NPI}$ следния клас задачи

$$\mathcal{NPI} = \mathcal{NP} \setminus (P \cup \mathcal{NPC}).$$

И така трите класа задачи са

Клас $\mathcal{NP}$



като сложността расте от "ляво на дясно". В класа $\mathcal{NPI}$ са задачите, за които не е намерен алгоритъм с полиномна сложност, но и не е доказано, че са $\mathcal{NP}$-пълни. До 1979 г. такава е била задачата на линейното оптимиране. След като Леонид Хачиян намира елипсоидален алгоритъм с полиномна сложност, който решава тази задача, тя влиза в класа $\mathcal{P}$ (вж. [6] и др.).

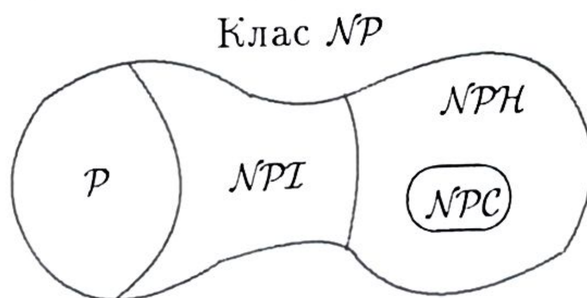
Класовете $\mathcal{NP}$ и $\mathcal{NPC}$ се отнасят за разпознавателни задачи. Като отслабим дефиницията за $\mathcal{NPC}$, ще получим класа на така наречените $\mathcal{NP}$ – hard ($\mathcal{NPH}$) задачи. Класът на $\mathcal{NP}$-трудните задачи се дефинира така

$$\mathcal{NPH} = \{P \mid \forall Q \in \mathcal{NP},\ Q \propto P\}.$$

Ще отбележим, че задачите в $\mathcal{NPH}$ може и да не са разпознавателни и освен това очевидно

$$\mathcal{NPC} \subset \mathcal{NPH}.$$

Графично класовете се илюстрират така:



За да се докаже, че една задача P е $\mathcal{NP}$-пълна, достатъчно е да се докаже, че $P \in \mathcal{NP}$ и да се намери $\mathcal{NP}$-пълна задача Q, за която $Q \propto P$. Казаното следва от дефинициите и теорема 4.3. Ако не можем да докажем $P \in \mathcal{NP}$, но намерим $\mathcal{NP}$-пълна задача Q, за която $Q \propto P$ ще установим, че задачата P е $\mathcal{NP}$-трудна.

Макар да нямат строг, а само препоръчителен характер, ще направим следните бележки. При изследване на една задача P е необходимо да установим към кой клас на сложност тя принадлежи. Ако е от класа $\mathcal{P}$, трябва да се опитаме да подобрим оценката на сложността на най-добрия известен алгоритъм. Ако е от класа $\mathcal{NP}$, се търси алгоритъм с полиномна сложност, т.е. опитваме се да я причислим към класа $\mathcal{P}$ или да докажем, че е $\mathcal{NP}$-пълна.

Тъй като $\mathcal{NP}$-пълните задачи са трудно решими, често се практикува следното:

- търси се *вероятностен алгоритъм*. Тези алгоритми намират оптималното решение с някаква вероятност;

- решава се *частен случай на задачата*, който е с полиномна сложност;

- търси се алгоритъм с експоненциална сложност. При конкретни задачи с неголеми размери (вж. табл. 4.1), експоненциално сложните алгоритми работят добре. Симплекс-методът например, е алгоритъм с експоненциална сложност, който върши чудесна практическа работа;

- търсят се *евристични алгоритми.* От всички злини се избира най-малката — вместо никакво решение, се намира някакво, за което няма гаранции, че е оптималното, нито пък е известно колко се отклонява от оптималното.

2. Анализ на изчислителната сложност на някои алгоритми

При фиксиран изходен граф алгоритмите на Дийкстра, Флойд и Данциг, които бяха разгледани в предишния параграф на тази глава, анализът на сложността се прави сравнително лесно. Това се отнася за всички алгоритми, при които броят на изпълнимите операции е практически постоянен. При други, какъвто е например алгоритъмът на Форд, не може отнапред да се знае какъв е броят операции, които ще бъдат изпълнени. При алгоритъма на Форд не е ясно предварително, колко пъти ще се оцветява всеки връх — това става ясно след изпълнението на алгоритъма. В такива алгоритми се определя горна граница на възможния брой операции. (Често това е твърде завишена оценка в сравнение с действително изпълняваните операции.)

Ще направим оценки за сложността на алгоритмите от предишния параграф,като ще предполагаме, че за всяка операция (събиране, търсене на минимум, максимум и т.н.) се изисква единица време. Това представлява интерес най-малко заради въпроса, дали е разумно алгоритъма на Флойд да заменим с алгоритъма на Дийкстра — последователно повтаряйки го, като за начален взимаме всеки от върховете на графа.

СЛОЖНОСТ НА АЛГОРИТЪМА НА ДИЙКСТРА. На първа итерация се разглежат $(n-1)$ неоцветени върха. Това се осъществява с помощта на

$$d(x) = \min\{d(x),\ d(p) + c(p,x)\},$$

което означава, че се извършват $(n-1)$ операции събиране и $(n-1)$ операции търсене на минимум. Освен това се избира най-малкото измежду $n-1$ числа, т.е. извършват се още $n-1$ операции сравнение. По този начин установихме, че първата итерация на алгоритъма включва $3(n-1)$ операции. Аналогично втората итерация ще включва $3(n-2)$ операции, третата — $3(n-3)$ и т.н. общият брой операции в алгоритъма на Дийкстра ще бъде

$$\sum_{i=1}^{n} 3(n-i) = \frac{3n(n-1)}{2}.$$

Разбира се на всяка итерация, освен споменатите операции, се извършва и оцветяване на връх и дъга, което е свързано с допълнителни изчислителни загуби при изпълнението на програмата, съставена за реализация на алгоритъма. Минимизирането на тези загуби е разгледано в [27], [28] и [29].

И така за алгоритъма на Дийкстра, намиращ **НКП** между s и всеки друг връх на пълен, свързан граф, получихме оценка $O(1,5n^2)$.

	Complexity ALG (Дийкстра) $= \mathbf{O(1,5n^2)}$.

Разбира се, това е горна граница за броя операции при търсене на $(s-t)$ **НКП** и тя се достига, когато t е последният връх, който се оцветява.

СЛОЖНОСТ НА АЛГОРИТЪМА НА ФОРД. В алгоритъм на Дийкстра всеки връх се оцветява веднъж, а в алгоритъма на Форд (модификация на Дийкстра) всеки връх може да бъде оцветяван до $(n-1)$ пъти. От това и като вземем предвид гореполучената оценка за сложността на алгоритъма на Дийкстра — $O(1,5n^2)$, получаваме за сложността на алгоритъма на Форд следната оценка:

	Complexity ALG (Форд) $= \mathbf{O(1,5n^3)}$.

СЛОЖНОСТ НА АЛГОРИТЪМА НА ФЛОЙД. В този алгоритъм се изчисляват n на брой матрици D^1, $D^2, \ldots,$ D^n, всяка от които е с размери $n \times n$, т.е. има n^2 елементи. Следователно се изчисляват n^3 елементи. Изчисляването се извършва с помощта на (3.5), а именно:

$$d_{ij}^m = \min\{d_{ij}^{m-1},\ d_{im}^{m-1} + d_{mj}^{m-1}\},$$

което изисква извършване на една операция събиране и една операция определяне на минимум. Следователно в алгоритъма на Флойд се изпълняват n^3 събирания и n^3 сравнения (определяне на минимум). Разбира се, тези числа силно надхвърлят действително извършваните в алгоритъма операции — да си припомним, че за всяко m елементите от m-тия ред и m-тия стълб се взимат от предишната матрица D^{m-1}, както и това, че диагоналните елементи също не се пресмятат, а са нули. И така количеството операции в алгоритъма на Флойд е пропорционално на $2n^3$ или с други думи

	Complexity ALG (Флойд) = O(2n³).

СЛОЖНОСТ НА АЛГОРИТЪМА НА ДАНЦИГ. Сложността на този алгоритъм е същата, както сложността на алгоритъма на Флойд, поради това, че в него се извършват същите операции, но в друг ред. Наистина (вж. (3.7) — (3.10) от предишния параграф):

- (3.7) от алгоритъма на Данциг се осъществява и в алгоритъма на Флойд;

- (3.8) от алгоритъма на Данциг съвпада с (3.5) от алгоритъма на Флойд;

- (3.9) и (3.10) от алгоритъма на Данциг просто $(m-1)$ пъти повтарят (3.5) от алгоритъма на Флойд.

От казаното следва, че двата алгоритъма включват еднакъв брой операции, т.е.

Complexity ALG (Данциг)= O(2n³).

СЛОЖНОСТ НА АЛГОРИТЪМА ЗА К-ТИ **НКП.** В този алгоритъм от изчислителна гледна точка най-обемна е *СТЪПКА 4*, където се извършват $O(n^2)$ операции, ако дъгите са с неотрицателни дължини или $O(n^3)$ операции, ако дъгите могат да са и с отрицателни дължини. Тъй като тази стъпка при k-тата итерация се изпълнява q_k пъти, а $q_k \geq n$ и броят на операциите е K, то алгоритъмът ще изпълнява брой операции от порядък Kn^3 или Kn^4.

Complexity ALG (К-ти НКП)= O(Kn³),

когато графът е с "неотрицателна" матрица на теглата.

Complexity ALG (К-ти НКП)= O(Kn⁴),

когато графът е с произволна матрица на теглата.

Ще направим едно уточнение на базата на резултат, получен от Д. Кнут. Очевидно редът на k-тите по дължина пътища не се променя, ако всички тегла d_{ij} заменим с $d'_{ij} = d_{ij} + h_i - h_j$, където h са произволни числа, приписани на върховете на графа. Използвайки този факт, Д. Кнут е показал, че взимайки $h_i =$ разстоянието (s, x_i), винаги можем да получим $d'_{ij} \leq 0$. Тъй като разстоянията (s, x_i) се получават в общия случай с помощта на $O(n^3)$ операции, то преобразувайки предварително теглата и в общия случай е възможно да се намерят K-ти **НКП** с използването на $O(Kn^3)$ операции.

КОМЕНТАР. 1. Ако в изходния граф няма дъги с отрицателни дължини, последователното (n-кратното) прилагане на алгоритъма на Дийкстра, като в качеството на начален се взема всеки от върховете на графа, ще води до $O(1,5n^3)$ операции.

Тази оценка е по-добра от порядъка $O(2n^3)$ на алгоритъма на Флойд*.

2. Ако обаче в изходния граф съществуват дъги с отрицателни дължини (без да съществуват цикли с отрицателна дължина), се налага вместо Дийкстра, да се ползва алгоритъмът на Форд. Тогава n-кратното прилагане на алгоритъма на Форд води до разход на време от порядъка на $O(1,5n^4)$. Това очевидно превишава оценката $O(2n^3)$ за алгоритъма на Флойд, т.е. в случая прилагането на алгоритъма на Флойд е рационално и ефективно. (Не забравяйте обаче, че оценката за алгоритъма на Форд е твърде завишена на практика).

2.5. Потоци в мрежи

Най-общо *потокът* определя начин за пренос на обекти, количества от един пункт в друг. Много често в редица транспортни задачи съществува пълна забрана за комуникация между отделните обекти или частична забрана, свързана с капацитета (пропускателната способност) на комуникациите — пътища, мостове, трасета и т.н. Най-често се иска максимизация (минимизация) на общия обем "превози" в системата, както и минимизация на цената (разходите за осъществяване на тези превози).

На езика на графите, потокът задава начин за пренос на обекти от един връх на графа в друг по неговите дъги (или ребра, ако графът е неориентиран). Началният връх, от който започва този пренос на количества, се нарича *източник* и обикновено се обозначава със s.

Върхът, до който трябва да бъде осъществен този пренос, се нарича *сток* (краен пункт, потребител, склад — stock, sink). Стокът се бележи обикновено с t.

Обектите, които се преместват, "протичат" от източника до стока, се наричат *единици на потока* или само *единици*.

Количеството единици на потока, което може да премине през дъгата (i,j), се нарича *капацитет (пропускателна способност) на дъгата. Максималната* пропускателна способност на една дъга (i,j) ще бележим със $c(i,j)$. По-нататък $c(i,j)$ ще наричаме просто пропускателна способност или капацитет.

---

*На практика обаче (поради това, че не всички операции изискват единица време и др.) прилагането на алгоритъма на Флойд, за графи с неотрицателна матрица на теглата, икономисва около 50% от времето в сравнение с n-кратното прилагане на алгоритъма на Дийкстра [30].

С $a(i,j)$ ще обозначаваме *цената (стойността) на преместването на единица от потока по дъгата* (i,j).

Да припомним, че *мрежа* — това е граф $G = (V, E)$, в който на всяка дъга е приписана някаква пропускателна способност (капацитет). Количеството единици, протичащи по дъгата (i,j), ще наричаме *поток в дадената дъга* и ще го обозначаваме с $f(i,j)$. В резюме:

Потокът f в една мрежа представлява функция, съпоставяща на всяка дъга (ребро) $e = (x, y)$ неотрицателно реално число $f(e) = f(x, y)$, така че:

За всеки поток от s в t, количеството единици, изходящи от произволен връх x, $x \neq s$, $x \neq t$, трябва да бъде равно на количеството единици, влизащи (входящи) в този връх x.

$$\sum_{y \in V} f(x,y) - \sum_{y \in V} f(y,x) = 0, \; x \neq s, \; t. \tag{5.1}$$

Освен това количеството единици на потока, проходящи по дъгата (x, y), не трябва да превишава нейния капацитет, т.е.

$$0 \leq f(x,y) \leq c(x,y), \; (x,y) \in E. \tag{5.2}$$

Накрая, общото количество единици на потока v, излизащо от източника s, трябва да бъде равно на сумарното количество единици на потока, влизащо в стока t.

$$\sum_{y \in V} f(s,y) - \sum_{y \in V} f(y,s) = v. \tag{5.3}$$

$$\sum_{y \in V} f(y,t) - \sum_{y \in V} f(t,y) = v. \tag{5.4}$$

И така $f(x, y)$, при $(x, y) \in E$ се нарича поток тогава и само тогава, когато са удовлетворени (5.1) — (5.4)*. Величината

*На други места в литературата, когато $f(x,y) \geq 0$, потокът се нарича *аритметичен*. В задачи, при които това изискване не е задължително, се дефинира понятието *алгебричен поток* (вж. [4]). Освен това се налага изискването източникът и стокът да бъдат единствени, като източникът само отдава количества, а стокът — само приема (вж. [3]).

на потока f се бележи с $val\ (f)$ или просто v и се определя по следния начин:

$$val\ (f) = \sum_{\forall y} f(s, y).$$

Задачата за намиране на *максимален поток* се състои в търсене на поток, удовлетворяващ горните изисквания (5.1) — (5.4), за който величината v е максимална. Очевидно задачата за максимизиране на v при ограниченията (5.1) — (5.4) е задача на линейното оптимиране и като такава тя може да се реши със симплекс-метода. Това обаче е твърде неефективно и ще наподобява "стрелба по врабче с оръдие" [1]. В този параграф ще разгледме един ефективен алгоритъм за търсене на максимален поток — *алгоритъм на Форд-Фалкерсон [10]*.

В зависимост от потока $f(x, y)$, дъгите на изходния граф (мрежата), можем да разделим условно на три категории:

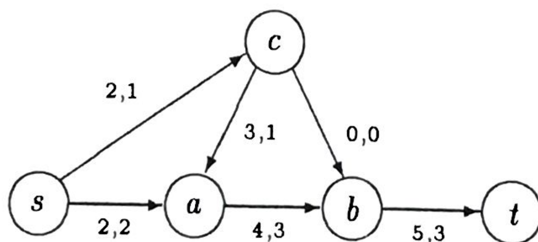
а) *дъги, в които потокът не може нито да се увеличава, нито да се намалява.* Множеството от тези дъги ще бележим с N. Това са дъгите, които имат нулев капацитет или "огромна" цена за преминаване на потока;

б) *дъги, в които потокът може да бъде намален.* Множеството от тези дъги се обозначава с R. Дъгите от множеството R се наричат *намаляващи*;

в) *дъги, в които потокът може да се увеличава.* Множеството от тези дъги се бележи с I. Дъгите от множеството I се наричат *увеличаващи (ненаситени)*.

Ясно е, че всяка дъга на графа принадлежи поне на едно от тези три множества N, R или I. Освен това $R \cap I \neq \emptyset$, в общия случай.

ПРИМЕР 5.1.



В зададената чрез горния граф мрежа, първите числа са капацитетите (пропускателните способности) на дъгите, а вторите числа са потока в съответната дъга. От източника s излизат три единици, колкото са единиците, постъпващи в стока t — изпълнени са (5.3) и (5.4). Очевидно е изпълнено и (5.2) — потокът не надхвърля капацитета на дъгата. За всеки връх, различен от s и t е изпълнено (5.1) — условието за съхраняване на потока. С други думи, в пример 5.1 сме задали поток f от s към t. За дъгите на графа в конкретния случай е изпълнено

$$(s,a) \in R; \quad (s,c) \in I,\ R; \quad (a,b) \in I,\ R;$$

$$(c,a) \in I,\ R; \quad (c,b) \in N; \quad (b,t) \in R,\ I.$$

Въпрос: Максимален ли е потокът в пример 5.1, т.е. има ли поток с величина v, такъв, че $v \geq 3$?

Отг.: Не е максимален, тъй като по дъгите (s,c), (c,a), (a,b) и (b,t) може да протече още една единица. Полученият по този начин поток ще удовлетворява (5.1) — (5.4) и общото количество единици v на потока от s към t ще бъде 4.

1. Алгоритъм за търсене на увеличаваща (потока) верига

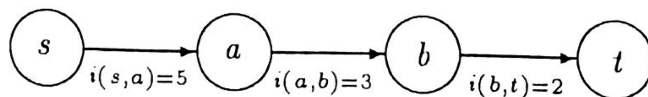
ИДЕЯ НА АЛГОРИТЪМА. Да обозначим с $r(x,y)$ максималното количество, с което може да бъде намален потокът (x,y), а с $i(x,y)$ максималната величина, с която може да се увеличи потокът в дъгата (x,y). Очевидно

$$r(x,y) = f(x,y) \quad \text{и} \quad i(x,y) = c(x,y) - f(x,y).$$

Какви са начините, за да протекат допълнително количество единици на потока от източника s в стока t?

а) Един такъв начин е да се намери път P от върха s до върха t, изцяло състоящ се от ненаситени (увеличаващи) дъги на множеството I.

ПРИМЕР 5.2.



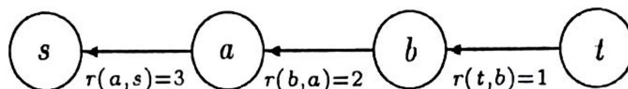
Тъй като $i(x, y)$ е максимално възможното увеличение на потока в дъгата (x, y), то величината на допълнителния поток от s към t в горния път P очевидно е равна на

$$\min_{(x,y)\in P}\{i(x, y)\}, \text{ т.е.}$$

$$\min\{5,\ 3,\ 2\} = 2;$$

б) Друг начин за увеличаване на потока е да се намери път P от стока t към източника s, изцяло състоящ се от дъги на множеството R, т.е. намаляващи дъги.

ПРИМЕР 5.3.



Максималната величина, с която може да се намали потокът във всяка дъга (x, y) на пътя P е $r(x, y)$, поради което максималното намаляване на потока по пътя P се определя като

$$\min_{(x,y)\in P}\{r(x, y)\}, \text{ т.е.}$$

$$\min\{1,\ 2,\ 3\} = 1.$$

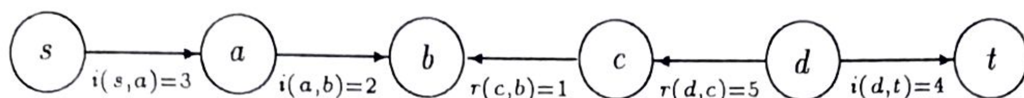
При това намаляването на потока във всяка дъга (x, y) води до намаляване на потока от стока t към източника s, което води до увеличение на чистия поток от върха s към върха t;

в) Комбинирането на горните два начина а) и б) за увеличение на чистия поток от s до t също е възможен. За целта трябва да се намери верига, съединяваща s и t, дъгите на която удовлетворяват следните две условия:

- всички *прави дъги* с направление от s към t са от I, т.е. са ненаситени $(c(x, y) - f(x, y) > 0)$;

- всички *обратни дъги*, с направление от t към s, принадлежат на множеството R, т.е. $f(x, y) > 0$.

ПРИМЕР 5.4. Да разгледаме следната верига, съединяваща s и t.



Тук правите дъги (s, a), (a, b) и (d, t) са ненаситени (увеличаващи), а обратните дъги (d, c) и (c, b) са от множеството R. Ако увеличим потока в правите дъги и намалим потока в обратните дъги, това ще доведе до възможността да формираме допълнителен поток от s в t по веригата. Ясно е, че величината на допълнителния поток по веригата се определя като минимум от следните две величини

$$\min\{i(x, y), \text{ където } (x, y) \text{ е права дъга}\},$$

$$\min\{r(x, y), \text{ където } (x, y) \text{ е обратна дъга}\}.$$

В случая $\min\{\min(3, 2, 4), \min(5, 1)\} = \min\{2, 1\} = 1$. Този минимум се нарича *максимално увеличение на потока по веригата* и в случая е 1.

По този начин увеличението на потока в правите дъги с 1 и намаляването на потока в обратните дъги също с толкова, дава възможност допълнително по веригата да се осъществи пренос от s към t на една единица поток.

ИДЕЯ НА АЛГОРИТЪМА. Идеята на алгоритъма за намиране на увеличаваща потока верига се състои в построяването на "растящо" от върха s дърво, състоящо се от оцветени дъги, по което от s могат да се изпратят допълнителни единици поток. Два са случаите, които възникват при изпълнение на алгоритъма:

- стокът t е оцветен — в този случай в построеното дърво единствената верига от оцветени дъги, свързваща s и t, се явява увеличаваща верига;

- в противен случай, т.е. t не е оцветен връх — не съществува увеличаваща верига между s и t. Ще опишем формално алгоритъма.

ОПИСАНИЕ НА АЛГОРИТЪМА ЗА ТЪРСЕНЕ НА УВЕЛИЧАВАЩА (ПОТОКА) ВЕРИГА. *СТЪПКА 1.* Определете състава на множествата N, I, R. Дъгите от множеството N изключете от разглеждане — в тях изменението на потока е невъзможно. Всички върхове са неоцветени. Оцветете върха s.

СТЪПКА 2. Оцветявайте дъгите (x, y) и върховете y на графа при вече оцветен връх x и неоцветен връх y по следния начин:

а) ако $(x, y) \in I$, оцветете дъгата (x, y) и върха y (*право оцветяване*);

б) ако $(y, x) \in R$, оцветете дъгата (y, x) и върха y (*обратно оцветяване*);

в) ако не сте в а) или б), не оцветявайте.

СТЪПКА 3. Ако след изпълнение на *стъпка 2* (т.е. процедурата за оцветяване), върхът t не е оцветен — край, в мрежата отсъства увеличаваща верига. В противен случай, т.е. върхът t е оцветен — в мрежата съществува единствена верига между s и t от оцветени дъги и тя е увеличаваща. Край.

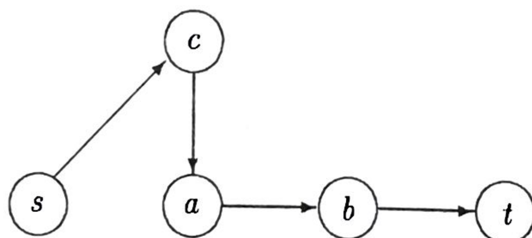
Ще илюстрираме алгоритъма със следния пример.

ПРИМЕР 5.5. Да разгледаме мрежата от пример 5.1 и за нея да приложим алгоритъма за търсене на увеличаваща верига.

СТЪПКА 1. Изпълнена е вече в пример 5.1. Оцветяваме върха s.

СТЪПКА 2. Тъй като s е оцветен и $(s, c) \in I$, оцветяваме дъгата (s, c) и върха c. Тъй като вече c е оцветен и $(c, a) \in I$, оцветяваме дъгата (c, a) и върха a. Тъй като a е оцветен и $(a, b) \in I$ — оцветяваме дъгата (a, b) и върха b. Тъй като b е оцветен и $(b, t) \in I$ — оцветяваме дъгата (b, t) и върха t.

СТЪПКА 3. Върхът t е оцветен. Край. Увеличаващата верига е



По нея, както вече казахме, може да протече още една единица, т.е. максималното увеличение на потока по веригата е 1.

ОБОСНОВКА НА АЛГОРИТЪМА. 1. Като се вземе предвид процедурата (начинът) за оцветяване — *СТЪПКА 2* а) и б), е ясно, че една дъга се оцветява само когато единият ѝ край е оцветен, а другият — не. При изпълнението на алгоритъма не е възможно да се оцвети дъга, на която и двата края вече са оцветени — това гарантира липсата на цикли, образувани от оцветени дъги. Тъй като първият оцветен връх е S, то алгоритъмът строи дърво, включващо s. Ако един връх x е оцветен, то ще съществува единствена верига от s до x от оцветени дъги. В частност, ако $x = t$, ще съществува верига, свързваща s и t. Веригата е увеличаваща поради това, че правите ѝ дъги са увеличаващи, а обратните — намаляващи.

2. Ако има увеличаваща верига от s в t, то очевидно върхът t трябва да бъде оцветен, и обратно — ако върхът t е оцветен, то съществува увеличаваща потока верига от s в t.

3. Предложената процедура свършва след краен брой стъпки по простата причина, че дъгите и върховете се оцветяват най-много по веднъж, а те са краен брой.

Алгоритъмът за търсене на увеличаваща потока верига не дава предписания за реда, в който трябва да се оцветяват върховете и дъгите (след като оцветим върха s). Това се извършва в произволен ред или като се изхожда винаги от последния оцветен връх. Намирането на увеличаваща верига дава възможност от източника да се пренесат допълнителни количества от потока до t, без разбира се да се надхвърля максималното увеличение на потока по веригата. Обърнете внимание, че в увеличаващите дъги на намерената верига потокът нараства, а в намаляващите дъги намалява с толкова, колкото е максималното увеличение на потока.

Малко по-късно, когато разглеждаме алгоритъма за търсене на максимален поток, ще дадем начин, по който може да се организира на практика процедурата за оцветяване на дъгите и върховете (при работа с компютър понятието оцветяване трябва да се "облече" в конкретни форми, подходящи за съхранение и организация в машината).

2. Алгоритъм за търсене на максимален поток (Алгоритъм на Форд-Фалкерсон)

[10] Идеята на алгоритъма се базира на многократно изпълнение на алгоритъма за търсене на увеличаваща потока верига, докато това е възможно. Ще дадем формално описание на този алгоритъм, като в него ще предложим и начин за оцветяване

(маркиране) на върховете. Този начин дава възможност след изпълнение на алгоритъма да разполагаме с дъгите, по които протича потокът, както и величината на потока от s в t.

ОПИСАНИЕ НА АЛГОРИТЪМА НА ФОРД—ФАЛКЕРСОН [10]

СТЪПКА 1. Изберете произволен поток от s до t в мрежата. Ако такъв няма, задайте "нулев" поток, т.е. за всяка дъга (x, y), $f(x, y) = 0$.

СТЪПКА 2. (първи етап — *процедура за оцветяване*) Оцветете върха s с наредената двойка $(-, \infty)$.

СТЪПКА 3. Ако съществува неоцветен връх y, при оцветен връх x:

а) Ако $(x, y) \in I$, т.е. $f(x, y) < c(x, y)$ — оцветете дъгата (x, y) и маркирайте (оцветете) върха y с двойката (x^+, Δ_y), където

$$\Delta_y = \min\{\Delta_x,\ c(x, y) - f(x, y)\} \quad \textit{(право маркиране)};$$

б) Ако $(y, x) \in R$, т.е. $f(y, x) > 0$ — оцветете дъгата (y, x) и маркирайте (оцветете) върха y с двойката (x^-, Δ_y), където

$$\Delta_y = \min\{\Delta_x,\ f(y, x)\} \quad \textit{(обратно маркиране)}.$$

След а) и б), преминете към *стъпка 4.* В противен случай преминете към *стъпка 7.*

СТЪПКА 4. Ако $y = t$, преминете към *стъпка 5* (край на първи етап). В противен случай се върнете на *стъпка 3.*

СТЪПКА 5. (втори етап — *формиране на нов поток*) Нека върхът y е маркиран с двойката (d_y, Δ_y). Тогава:

а) Ако $d_y = x^+$, положете $f(x, y) = f(x, y) + \Delta_t$;

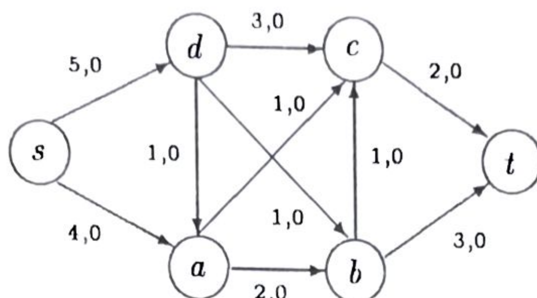
б) Ако $d_y = x^-$, положете $f(y, x) = f(y, x) - \Delta_t$.

СТЪПКА 6. Ако $x = s$, отстранете (изтрийте) всички маркировки (край на втори етап) и се върнете на *стъпка 2.* В противен случай положете $y = x$ и се върнете на *стъпка 5.*

СТЪПКА 7. Край. Полученият поток е максимален.

Да илюстрираме работата на алгоритъма със следния пример.

ПРИМЕР 5.6.

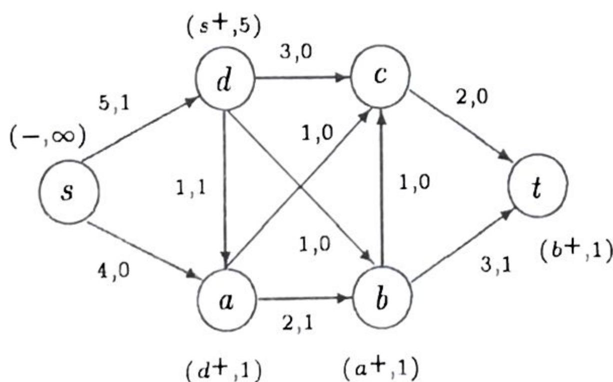


СТЪПКА 1. Разглеждаме нулевия поток f_o от s до t, т.е. $f_0(x, y) = 0$, за всяка дъга от мрежата. Първите числа, приписани на всяка от дъгите в мрежата, е съответният капацитет на дъгата.

СТЪПКА 2. Маркираме (оцветяваме) върха s с двойката $(-, \infty)$. Изпълняваме неколкократно *СТЪПКА 3.* Последователно

- оцветяваме дъгата (s, d) и маркираме върха d с двойката $(s^+, \min(\infty, 5 - 0)$, т.е. $(s^+, 5)$;
- оцветяваме дъгата (d, a) и маркираме (оцветяваме) върха a с двойката $(d^+, \min(5, 1 - 0)) = (d^+, 1)$;
- оцветяваме дъгата (a, b) и маркираме (оцветяваме) върха b с двойката $(a^+, 1)$;
- оцветяваме дъгата (b, t) и маркираме върха t с двойката $(b^+, 1)$.

Тъй като върхът t е оцветен, преминаваме към *СТЪПКА 5.* Изпълнявайки *СТЪПКА 5*, ще получим (първият символ в наредената двойка е указател за върха, от който y получава маркировката и за вида маркиране — право или обратно):



Получаваме нов поток f_1 от s до t, при който

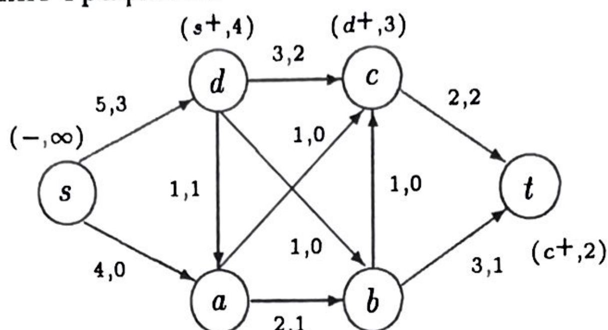
$$
\begin{aligned}
f_1(b, t) &= f_0(b, t) \quad +\Delta_t = 0 + 1 = 1.\\
f_1(a, b) &= f_0(a, b) \quad +\Delta_t = 0 + 1 = 1.\\
f_1(d, a) &= f_0(d, a) \quad +\Delta_t = 0 + 1 = 1.\\
f_1(s, d) &= f_0(s, d) \quad +\Delta_t = 0 + 1 = 1.
\end{aligned}
$$

Формирахме поток f_1 от s до t, за който

$$val\ (f_1) = 1 = \sum_{\forall j} f(s, j)$$

и отстраняваме досегашните маркировки на върховете. Връщаме се на *СТЪП-КА 2.*

Изпълнявайки алгоритъма последователно, ще получим следните резултати, дадени за улеснение графично.



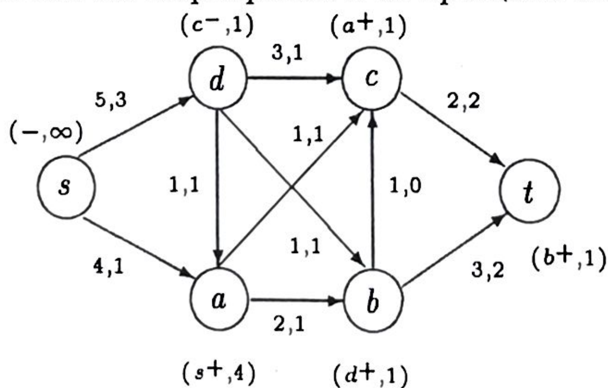
Намерихме увеличаваща за потока f_1 верига

$$(c, t),\ (d, c) \text{ и } (s, d).$$

За новия поток f_2, даден с графа по-горе, имаме

$$val\ (f_2) = 3.$$

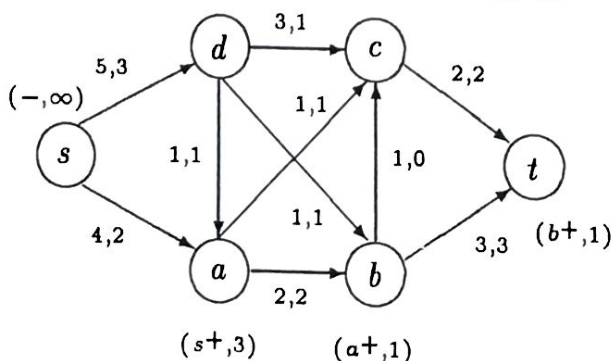
Отново анулираме всички маркировки и се връщаме на *СТЪПКА 2.*



За новия поток f_3 имаме

$$val\ (f_3) = \sum_{\forall j} f(s, j) = 4.$$

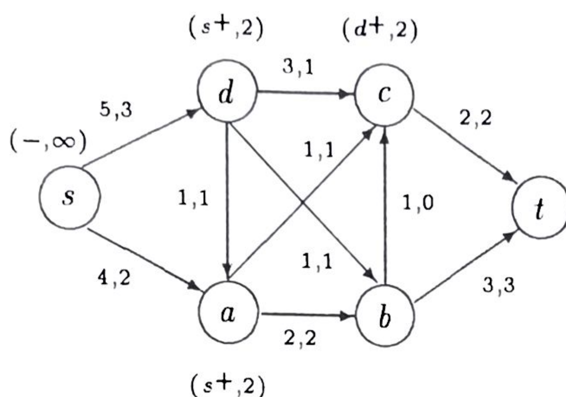
Анулираме маркировките и се връщаме на *СТЪПКА 2.*



За получения нов поток f_4 имаме

$$val\ (f_4) = 5.$$

Това е и максималният поток от s до t, тъй като връщането към *СТЪПКА 2* води до



Черт. 2.2

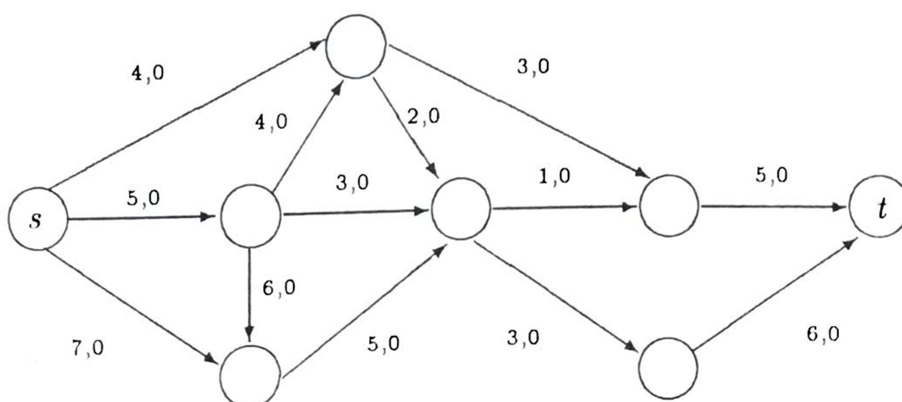
Няма други върхове, които могат да се маркират. Край на алгоритъма. Потокът f_4 е максимален.

ЗАДАЧА 5.1. Единствен ли е максималният поток за мрежата от пример 5.6?

Отг.: Не. Ако променим получения поток по следния начин: $f(s,d) = 2$, $f(s,a) = 3$, $f(d,a) = 0$, ще получим поток, различен от предишния, чиято величина също е 5.

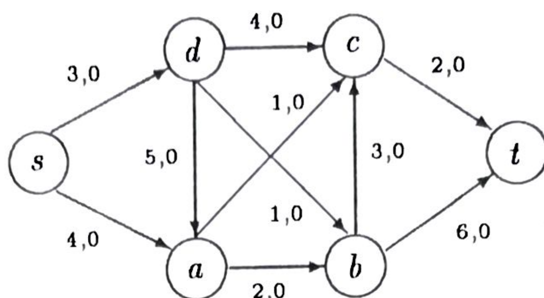
ЗАДАЧА 5.2. За дадените по-долу мрежи да се определят максималните потоци, като се използува методът на Форд-Фалкерсон:

а)



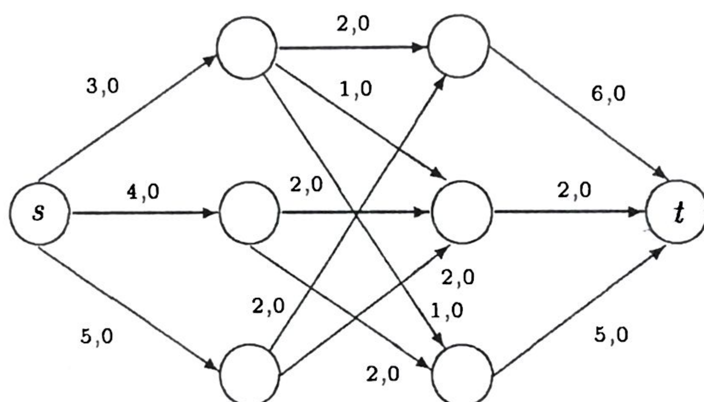
Отг.: Максимален поток f, $val\ (f) = 7$.

б)



Отг.: Максимален поток f, $val\ (f) = 5$.

в)



Отг.: Максимален поток f, $val\ (f) = 9$.

ОБОСНОВКА НА АЛГОРИТЪМА НА ФОРД-ФАЛКЕРСОН.

Теорема за максималния поток и минималния разрез.

Да припомним определението на понятието разрез. Нека $G = (V, E)$ е произволен свързан граф. Ако $T \cup \overline{T} = V$ и $T \cap \overline{T} = \emptyset$, множеството от всички дъги, на които единият връх е в T, а другият — в $\overline{T}$, се нарича разрез на графа G. Бележи се $\langle T, \overline{T} \rangle$.

Ще казваме, че $\langle T, \overline{T} \rangle$ разделя източника s и стока t, ако $s \in T$, а $t \in \overline{T}$. Такъв разрез ще наричаме $(s - t)$-*разрез.*

Пропускателната способност (капацитетът) $c(K) = c(T, \overline{T})$ на разреза $K = \langle T, \overline{T} \rangle$, се определя като

$$c(K) = \sum_{x \in T,\ y \in \overline{T}} c(x, y). \tag{5.5}$$

Очевидно е, че пропускателната способност на дъгите, ориентирани от $\overline{T}$ към T не увеличава пропускателната способност на разреза $K = \langle T, \overline{T} \rangle$. Да означим с $f(T, \overline{T})$ сумата от потоците в дъгите, ориентирани от T в $\overline{T}$, аналогично с $f(\overline{T}, T)$ сумата от потоците в дъгите, ориентирани от $\overline{T}$ в T.

ЗАДАЧА 5.3. Да се определи капацитетът на разреза $K = \langle T, \overline{T} \rangle$ на графа от черт. 2.2 (пример 5.6), където

$$T = \{s, \ a, \ d, \ c\} \ \text{ и } \ \overline{T} = \{b, \ t\}.$$

Решение: Дъгите, ориентирани от T в $\overline{T}$, са:

$$(a, b), \ (d, b), \ (c, t).$$

Следователно $c(T, \overline{T}) = c(a, b) + c(d, b) + c(c, t)$, т.е.

$$c(K) = c(T, \overline{T}) = 2 + 1 + 2 = 5.$$

▷ *ТЕОРЕМА 5.1. За всеки поток f и всеки $(s - t)$-разрез $\langle T, \overline{T} \rangle$ в дадена мрежа,*

$$val\ (f) = f(T, \overline{T}) - f(\overline{T}, T). \tag{5.6}$$

Доказателство: От дадените определения за поток имаме

$$\sum_{\forall y} f(x, y) - \sum_{\forall y} f(y, x) = \begin{cases} val\ (f), & \text{при } x = s, \\ 0, & \text{при } x \in T \setminus \{s\}. \end{cases} \tag{5.7}$$

Сумирайки горните равенства по всички върхове от T, ще получим

$$\sum_{x \in T} \sum_{\forall y} f(x, y) - \sum_{x \in T} \sum_{\forall y} f(y, x) = val\ (f). \tag{5.8}$$

В лявата част на горното равенство $f(x, y)$ и $-f(x, y)$, при $x \in T$ и $y \in T$, се появяват точно по веднъж и взаимно се унищожават. Поради това от (5.8) следва

$$\sum_{x \in T} \sum_{y \in \overline{T}} f(x,y) - \sum_{x \in T} \sum_{y \in \overline{T}} f(y,x) = val\ (f), \quad \text{т.е.}$$

$$val\ (f) = f(T, \overline{T}) - f(\overline{T}, T),$$

което трябваше да се докаже. ◁

Обърнете внимание, че $val\ (f) = \sum_{\forall y} f(s,y)$ се явява частен случай на равенството (5.6).

СЛЕДСТВИЕ 1. За всеки поток f и всеки $(s-t)$-разрез $K = \langle T, \overline{T} \rangle$ в мрежата, е изпълнено

$$val\ (f) \leq c(T, \overline{T}). \tag{5.9}$$

(В частност, величината на максималния поток е не по-голяма от капацитета на всеки разрез — включително и този с най-малък капацитет). **Доказателство:** Тъй като $f(x,y) \geq 0$, от теоремата следва

$$val\ (f) = f(T, \overline{T}) - f(\overline{T}, T) \leq f(T, \overline{T}) \leq c(T, \overline{T}). \tag{5.10}$$

◁

Да припомним, че при $f(x,y) < c(x,y)$, дъгата се нарича *f-ненаситена*, а при $f(x,y) = c(x,y)$ — *f-наситена*. Дъгите от R, т.е. онези, за които $f(x,y) > 0$, се наричат понякога *f-положителни*, а дъгите с "нулев" поток — *f-нулеви*.

Знакът равенство в (5.9) се достига тогава и само тогава, когато

$$f(\overline{T}, T) = 0 \text{ и } f(T, \overline{T}) = c(T, \overline{T}), \text{ т.е.}$$

$$val\ (f) = c(T, \overline{T})$$

тогава и само тогава, когато всички дъги, ориентирани от T в $\overline{T}$ са f-наситени, а дъгите, ориентирани от $\overline{T}$ в T са f-нулеви.

Разрезът $K_{\min} = \langle T, \overline{T} \rangle$, разделящ върховете s и t, ще наричаме *минимален разрез*, ако не съществува $(s-t)$-разрез K в мрежата, такъв, че $c(K) < c(K_{\min})$.

СЛЕДСТВИЕ 2. Ако f е поток и K е $(s-t)$-разрез, за който

$$val\ (f) = c(K),$$

то f е максимален поток, а K е минимален $(s-t)$-разрез.

Доказателство: Да означим максималния поток с $f_{\max}$ и минималния $(s-t)$-разрез с $K_{\min}$. От следствие 1 имаме

$$val\ (f_{\max}) \leq c(K_{\min}).$$

Следователно, за произволен поток f и произволен разрез K ще имаме

$$val\ (f) \leq val\ (f_{\max}) \leq c(K_{\min}) \leq c(K).$$

По условие $val\ (f) = c(K)$. Следователно

$$val\ (f) = val\ (f_{\max}) = c(K_{\min}) = c(K),$$

т.е. f е максимален поток, а K е минимален $(s-t)$-разрез. ◁

Вече можем да пристъпим към същината на обосновката на алгоритъма на Форд-Фалкерсон. Както изяснихме по-рано, потокът в мрежата може да се увеличи, ако се намери увеличаваща верига между s и t — верига, на която правите дъги са f-ненаситени, а обратните дъги са f-положителни.

▷ *ТЕОРЕМА 5.2. Потокът f в дадена мрежа е максимален тогава и само тогава, когато в мрежата няма увеличаваща верига, свързваща s и t.*

Доказателство: *Необходимост:* Нека потокът f е максимален. Ако допуснем, че в мрежата съществува увеличаваща верига, ще получим нов поток f^*, за който $val\ (f^*) > val\ (f) = val\ (f_{\max})$. Противоречие! Следователно в мрежата не съществува увеличаваща верига.

Достатъчност: Нека сега в мрежата не съществува увеличаваща $(s-t)$-верига. Да означим с T множеството от върхове в мрежата, до които съществуват увеличаващи вериги от източника s $(s \in T)$. Очевидно за стока t в нашия случай имаме

$t \in \overline{T} = V \setminus T$ (допускането на противното, т.е. $t \notin \overline{T}$ води до противоречие с условието да не съществуват увеличаващи $(s-t)$-вериги). Разрезът $\langle T, \overline{T} \rangle$ е $(s-t)$-разрез. Ще докажем, че

$$val\ (f) = c(T, \overline{T}).$$

Да разгледаме произволна дъга (x, y), за която $x \in T$, а $y \in \overline{T}$. От това, че $x \in T$, следва, че съществува увеличаваща $(s-x)$-верига. Дъгата (x, y) трябва да бъде f-наситена, тъй като в противен случай ще съществува увеличаваща $(s-y)$-верига, т.е. y трябва да бъде от множеството T. Аналогично всяка дъга (y, x), ориентирана от $\overline{T}$ в T, трябва да бъде f-нулева. Следователно

$$f(T, \overline{T}) = c(T, \overline{T}) \quad и \quad f(\overline{T}, T) = 0.$$

Тогава $val\ (f) = f(T, \overline{T}) - f(\overline{T}, T) = c(T, \overline{T})$. ◁

От второто следствие на предишната теорема следва, че f е максимален поток, а $\langle T, \overline{T} \rangle$ е минимален разрез. По този начин ние установихме следния добре известен резултат на Форд-Фалкерсон, а именно:

▷ *ТЕОРЕМА 5.3. Във всяка мрежа величината на максималния поток е равна на пропускателната способност (капацитета) на минималния разрез.* ◁

Ще отбележим, че задачата за намиране на минимален разрез е двойнствена (дуална) на задачата за максималния поток. Теоремата на Форд-Фалкерсон се явява аналог на теоремата за двойнственост при транспортна мрежа.

И така при последното маркиране на върховете, т.е. когато "алгоритъмът установи" липсата на увеличаваща верига, (т.е. върхът t не е маркиран и няма други върхове, които могат да бъдат маркирани) и спре своята работа, имаме:

1. Оцветените (маркираните) върхове образуват множество T, $s \in T$;

2. Неоцветените (немаркираните) върхове образуват множество $\overline{T} = V \setminus T$, $t \in \overline{T}$;

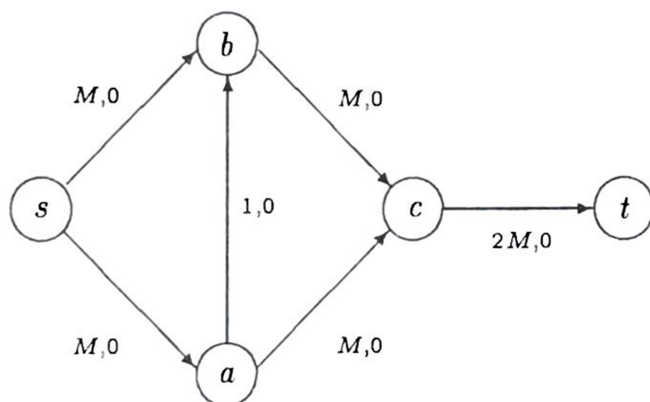
3. Разрезът $\langle T, \overline{T} \rangle$ е минимален и стойността на неговия капацитет (пропускателната способност) съвпада с $val\ (f_{\max})$ (вж. пример 5.6 и задача 5.3).

В разглежданията, които направихме дотук, не беше казано нищо по въпроса дали предложеният алгоритъм на Форд-Фалкерсон приключва работа след краен брой стъпки. Ще разгледаме сега и този въпрос.

3. Модификация на Едмондс и Карп

В алгоритъма на Форд-Фалкерсон, който беше описан, дъгите и върховете се маркираха в произволен порядък. С други думи, увеличаващата верига (когато съществува) се формираше по произволен начин. Това крие опасност алгоритъмът в общия случай да не завършва след краен брой стъпки.

ПРИМЕР 5.7. Да разгледаме следната мрежа



и приложим алгоритъма на Форд-Фалкерсон, последователно използвайки пътищата

$$P_1 : \ (s,a),\ (a,b),\ (b,c),\ (c,t);$$
$$P_2 : \ (s,b),\ (b,a),\ (a,c),\ (c,t).$$

Очевидно на всяка стъпка ще получаваме потоци, на които величината ще нараства с 1. Максималният поток, чиято величина е $2M$, ще се получи след $2M$ стъпки (нараствания на потока). Оказва се, че в този случай броят на стъпките, които се изпълняват, не се явява функция, зависеща от броя на върховете и ребрата на мрежата, а е функция на капацитета M, който в много задачи може да бъде произволно голям. Форд и Фалкерсон [10] са показали освен това, че алгоритъмът им е краен при изискването за целочисленост на потоците и капацитетите на дъгите. Те са показали, че алгоритъмът не намира

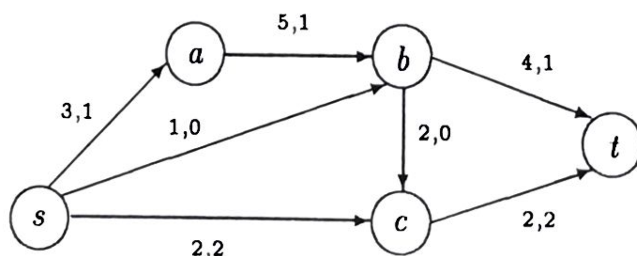
решение, ако пропускателните способности на дъгите са ирационални числа — приведен е пример, в който величината на потока клони към $\frac{1}{4}$ от величината на максималния поток при безкрайно нарастване броя на стъпките.

Едмондс и Карп в 1972 г. ([31]) усъвършенстват маркиращия алгоритъм. Тяхната модификация гарантира независимост на броя на стъпките, необходими за реализация на маркиращия алгоритъм, от пропускателните способности на дъгите.

ИДЕЯ НА АЛГОРИТЪМА НА ЕДМОНДС И КАРП. На всяка стъпка потокът се увеличава по възможно най-краткия път (верига). По-точно, увеличението на потока се извършва по вериги с най-малък брой дъги. Намирането на такива вериги става по правилото "първият маркиран е първият разгледан". За целта върхът s получава номер 1. След това се оцветяват дъгите, инцидентни с върха 1 (за които това е възможно), после се оцветяват (ако е възможно) дъгите, инцидентни с върха 2 и т.н. Оцветяването се извършва по правилата, дадени в алгоритъма на Форд-Фалкерсон. Тази процедура за оцветяване (маркиране) гарантира всяка верига от оцветени дъги, съединяваща източника s с произволен връх, да съдържа минимален брой дъги.

Освен това при тази процедура за маркиране, изискуемите стъпки за реализация на маркиращия алгоритъм са функция на броя върхове и ребра на мрежата (а не на пропускателните способности на дъгите).

ПРИМЕР 5.8. Да се приложи модифицираната процедура на Едмондс и Карп за оцветяване при търсене на максималния поток в следната мрежа:



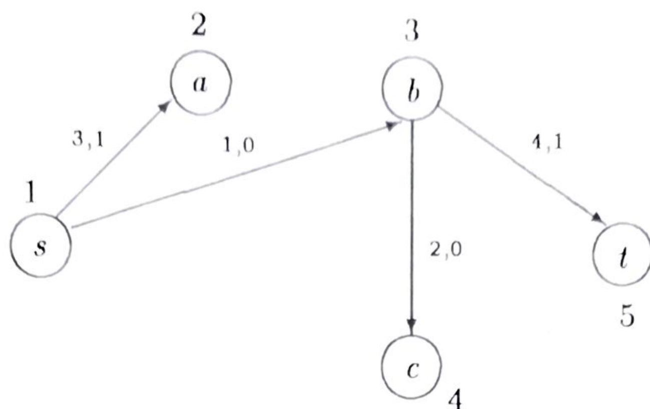
Оцветяваме върха s и го номерираме с 1.

След това разглеждаме неоцветените дъги, инцидентни със s и оцветяваме тези, за които това е възможно: Оцветяваме (s,a) и върха a. Върхът а получава номер 2; Оцветяваме (s,b) и върха b. Върхът b получава номер 3 (дъгата (s,c) е инцидентна със s, но тя не може да бъде оцветена поради $f(s,c) = 2 = c(s,c)$).

След това разглеждаме неоцветените дъги, инцидентни с връх 2, т.е. върха a: Няма дъга, инцидентна с връх a, която може да се оцвети (единствената дъга (a, b) е с оцветени крайни върхове).

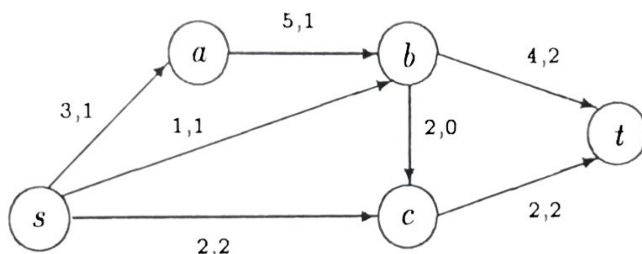
Разглеждаме неоцветените дъги, инцидентни с b: Оцветяваме (b, c) и върха c. На върха c приписваме номер 4; Оцветяваме (b, t) и върха t. Номерираме t с 5.

В полученото дърво с корен s



единствената увеличаваща $(s - t)$-верига е (s, b), (b, t), която е с минимален брой дъги и по нея потокът може да бъде увеличен с единица.

Изтриваме (снемаме) номерацията на всички върхове, оцветяванията на върховете и дъгите, след което повтаряме процедурата за мрежата (в нея вече е формиран нов поток).



Оцветяваме върха s и го номерираме с 1.

Разглеждаме неоцветените, инцидентни със s дъги и оцветяваме тези от тях, за които това е възможно:

Оцветяваме (s, a) и върха a. Номерираме върха a с 2. Дъгата (s, b) е инцидентна със s, но тя не може да бъде оцветена, тъй като $f(s, b) = 1 = c(s, b)$. Аналогично дъгата (s, c) също не може да бъде оцветена $\left(f(s, c) = 2 = c(s, c)\right)$.

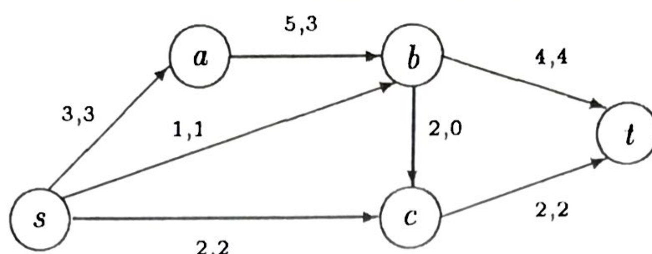
След това разглеждаме неоцветените дъги, инцидентни с a. Оцветяваме дъгата (a, b) и върха b, като му присвояваме номер 3.

Разглеждаме неоцветените дъги, инцидентни с връх 3, т.е. върха b. Оцветяваме дъгата (b, c) и върха c, като му присвояваме номер 4. Оцветяваме дъгата (b, t) и върха t, като го номерираме с 5.

Получихме увеличаваща верига с минимален брой ребра

$$(s, a), \ (a, b), \ (b, t),$$

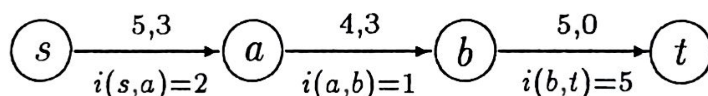
по която потокът може да бъде увеличен с две единици. Така получаваме следната мрежа:



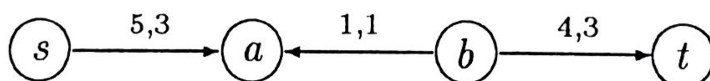
По-нататъшното увеличение на потока е невъзможно, което можете да установите, като приложите още веднъж предложената процедура от Едмондс и Карп. Максималният поток е с величина 6.

ОБОСНОВКА НА МОДИФИКАЦИЯТА НА ЕДМОНДС И КАРП.

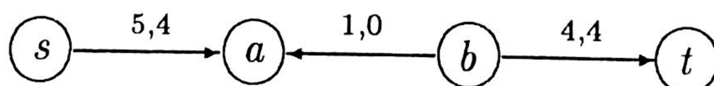
За целта ще разгледаме следното понятие. В една увеличаваща верига дъгата (x, y) се нарича *тясно място*, ако именно тази дъга ограничава увеличаването на потока. Например: В разгледаната по-долу увеличаваща верига, дъгата (a, b) е тясно място. Очевидно в произволна верига може да има повече от едно тесни места.



Максималното увеличение на потока по увеличаващата верига води до $f(x, y) = c(x, y)$ или $f(x, y) = 0$ за тесните места. Например:



Максималното увеличение на потока по веригата е единица. Това води до



Да пристъпим сега към намирането на една оценка за броя на нарастванията на потока, която ще бъде функция на броя на дъгите и ребрата в мрежата (графа).

За да оценим броя на увеличаващите вериги, които се формират от алгоритъма на Едмондс и Карп, ще разсъждаваме така: При всяко намиране на увеличаваща верига (нов по-голям поток), поне една дъга (x, y) се явява тясно място. В най-лошия случай, при всяко увеличение на потока само една дъга ще

се явява тясно място. Следователно трябва да отговорим на въпроса: "Ако дъгите в мрежата са m на брой, по колко пъти всяка от тези дъги може да се явява тясно място?" Ще докажем, че произволна дъга (x, y) не може да бъде тясно място в увеличаващите $(s - t)$ вериги повече от $\frac{n}{2}$ пъти, т.е. броят на увеличенията на потока е не повече от $\frac{m.n}{2}$.

Да предположим, че дъгата (x, y) се явява тясно място в две формирани от алгоритъма вериги C_1 и C_2. При това C_2 се формира след C_1 и тя е първата измежду формираните след C_1 вериги, в която (x, y) е тясно място. Без ограничения на общността, нека дъгата (x, y) се явява права дъга във веригата C_1, т.е. след увеличение на потока по веригата C_1, $f(x, y) = c(x, y)$. Очевидно за веригата C_2 дъгата (x, y) ще бъде обратна. Да означим с $B_i(p, q)$ броя на дъгите във веригата C_i между върховете p и q, $i = 1, 2$. Модификацията на Едмондс и Карп оцветява дъгите и върховете, така че построяваните от алгоритъма вериги с начало s, са с минимален брой дъги. От това следва верността на следните релации:

$$B_1(s, y) \leq B_2(s, y), \quad B_1(x, t) \leq B_2(x, t)$$

$$B_1(s, t) = B_1(s, y) + B_1(x, t) - 1 \leq$$
$$\leq B_2(s, y) + B_2(x, t) - 1 = B_2(s, t) - 2.$$

Следователно $B_1(s, t) + 2 \leq B_2(s, t)$, т.е. всеки път, когато (x, y) става тясно място, броят на дъгите в съответната увеличаваща потока верига нараства поне с 2.

Нека C_1, $C_2, \ldots$, C_k са всички вериги, за които (x, y) е тясно място. Тъй като всяка верига има не повече от $(n - 1)$ дъги, то

$$B_k(s, t) \leq n - 1 \Longrightarrow 1 + (k - 1).2 \leq n - 1,$$

откъдето $k \leq \frac{n}{2}$, т.е. броят на веригите, в които (x, y) е тясно място, е не повече от $\frac{n}{2}$. Да припомним, че броят на дъгите в мрежата е m, откъдето следва, че общият брой нараствания на потока не надхвърля числото $\frac{m.n}{2}$.

До същия резултат щяхме да стигнем, ако при промяна на потока във веригата C_1, вместо предположението, че потокът в дъгата (x, y) се увеличава до капацитета $c(x, y)$, считахме, че потокът в (x, y) се намалява до нула.

В горните разсъждения единственото наложено изискване беше неотрицателност на пропускателните способности на дъгите. От казаното следва, че използваният в алгоритъма на Форд-Фалкерсон принцип за оцветяване "първият оцветен е първият

разгледан", води до краен брой увеличения на потока при реални неотрицателни пропускателни способности.

Тъй като увеличаваща верига може да се намери след $O(m)$ стъпки, от получената по-горе оценка следва, че алгоритъмът има сложност $O(m^2.n)$. Има редица изменения, подобряващи ефективността от работата на алгоритъма в някои класове мрежи.

Интересен и ефективен метод със сложност $O(n^2)$ е предложен от Диниц в [48]. Методът се базира на построяване на спомагателна, ациклична мрежа, чиято структура точно определя всички най-кратки $(s - t)$ увеличаващи вериги за потока.

4. Алгоритъм за търсене на максимален поток при няколко източника и стока

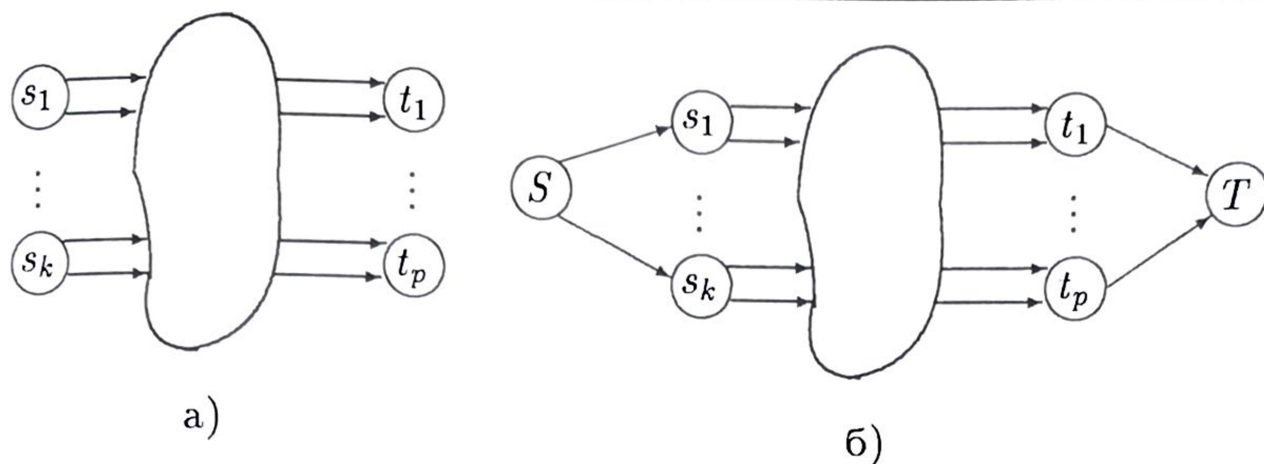
Досега разглеждахме мрежи с един източник и един сток. Алгоритъмът за търсене на максимален поток може да се приложи и в случаите, когато в изходната мрежа съществуват $s_1, s_2, \ldots, s_k$ източника и $t_1, t_2, \ldots, t_p$ стока. За целта е достатъчно мрежата да се допълни с един главен източник S и главен сток T. Главният източник S се съединява с всеки от източниците $s_1, s_2, \ldots, s_k$, чрез дъгите

$$(S, s_1),\ (S, s_2), \ldots,\ (S, s_k),$$

чиято пропускателна способност се приема за неограничена. Аналогично $t_1, t_2, \ldots, t_p$ се съединяват с главния сток T чрез дъгите

$$(t_1, T),\ (t_2, T), \ldots,\ (t_p, T),$$

чиято пропускателна способност се счита неограничена. По този начин изходната мрежа се разширява до следната мрежа - черт. 2.3 а) и б).



Черт. 2.3

Очевидно на всеки поток в разширения граф съответства поток от s_1, $s_2, \ldots$, s_k до t_1, $t_2, \ldots$, t_k в изгодния граф и обратно. При това на максималния поток в разширения граф съответства максималния поток в изходния граф, т.е. алгоритъмът за търсене на максимален поток може да се приложи в разширения граф (мрежа) и намереният поток ще определя максимален поток в изходния граф (мрежа).

5. Алгоритъм за търсене на поток с минимална стойност (Форд-Фалкерсон)

[10] В предишните параграфи разгледахме алгоритми за търсене на максимален поток — пренос на максимален брой единици (количества) от източника в стока по дъгите на мрежата. Сега ще си поставим за цел да организираме преноса на едно зададено количество от v единици поток между източника и стока.

С $a(x, y)$ ще означаваме *цената (стойността, разходите)* за преноса на 1 единица от потока по дъгата (x, y). Да припомним, че $f(x, y)$ означава количеството единици от потока, преминаващи по дъгата (x, y), $f(x, y) \geq 0$. Както преди, със $c(x, y)$ ще бележим капацитета на дъгата (x, y).

Очевидно решаването на много транспортни задачи (движение на количества — суровини, стоки, хора и т.н.) в мрежи, поставят проблема за търсене на *пренос с минимална цена*. За краткост, понякога *потокът с минимална стойност* ще наричаме *минимален поток*. Освен това засега ще предполагаме, че *цените* $a(x, y)$ са цели положителни числа. Това ограничение не е силно, поради факта, че цени от типа на "£5.25" могат да се разглеждат като "525 pence".

Съществува модификация на алгоритъма за търсене на по-

ток с минимална стойност, за който са допустими нецелочислени цени $a(x, y)$. Съществува и алгоритъм за търсене на поток с минимална стойност, работещ и за цени $a(x, y) < 0$ — *алгоритъм на дефекта*. В края на параграфа тези два алгоритъма ще бъдат също разгледани.

Задачата за търсене на минимален поток може да бъде представена по следния начин:

(5.11) $$\sum_{(x,y)} a(x, y).f(x, y) \to \min$$

при условия

(5.12) $$\sum_{y} f(s, y) - \sum_{y} f(y, s) = v,$$

(5.13) $$\sum_{y} f(t, y) - \sum_{y} f(y, t) = -v,$$

(5.14) $$\sum_{y} f(x, y) - \sum_{y} f(y, x) = 0, \ x \neq s, \ t,$$

(5.15) $$0 \leq f(x, y) \leq c(x, y), \text{ при } \forall x, \ \forall y.$$

Очевидно задачата за максимален поток може да се разглежда като частен случай на задачата за поток с минимална стойност, в който цените на дъгите са нула, а v съвпада с величината на максималния поток. Освен това (5.11) може да се замени с

(5.16) $$\{p - \sum_{(x,y)} a(x, y).f(x, y)\} \to \max,$$

където p е достатъчно голямо число (p — произволно число, по-голямо от максималната цена за пренос на единица поток от

източника s в стока t). Ако p се интерпретира като доход, получаван в резултат на $(s-t)$ пренос на единица от потока, (5.16) може да се интерпретира като максимална печалба. При тази интерпретация постигането на максимум в (5.16) ще води до минимум в (5.11) и обратно.

ИДЕЯ НА АЛГОРИТЪМА. Алгоритъмът за търсене на поток с минимална стойност "се опитва да пренесе" от s до t максимално възможния брой единици от потока, за всяка от които общата цена за пренос в мрежата е нула (лева). След това от източника s до стока t се пренасят колкото се може повече единици от потока, за всяка от които цената за пренос по мрежата е 1 (лев). На всяка стъпка, при изпълнение на алгоритъма общата цена за пренос на единица от потока по мрежата се увеличава с единица. Алгоритъмът завършва своята работа при сбъдване на някое от следните две събития:

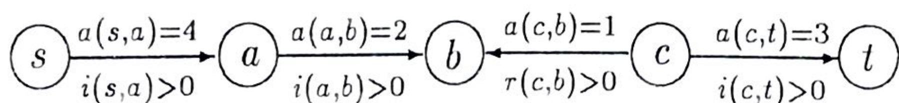
а) в мрежата не могат да се пренасят допълнителни единици поток;

б) в мрежата е осъществен пренос на дадените v единици поток.

С други думи, в алгоритъма за търсене на поток с минимална стойност многократно се решава задачата (5.12) — (5.16), при $p=0$, при $p=1$, $p=2$ и т.н.

Да предположим, че в изходната мрежа (началният поток е нулев) са пренесени (пропуснати) максимален брой единици от потока, за всяка от които стойността на преминаване през мрежата е $p-1$. Алгоритъмът реализира търсене на увеличаваща верига, по която могат да протекат нови единици от потока, стойността на всяка от които е p. Как става това?.

Да разгледаме следната увеличаваща потока $(s-t)$ верига.



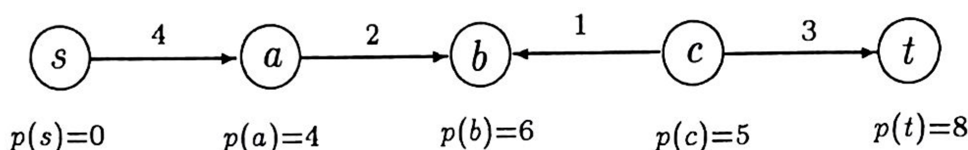
Алгоритъмът оцветява дъги и върхове, като на всеки връх x се съпоставя маркиращо число $p(x)$ по следния начин:

$$p(s)=0;\quad p(t)=p;\quad 0\le p(x)\le p,\ \text{при } x\ne s,\ t,$$
$$p(y)-p(x)=a(x,y),\ \text{при } (x,y)\in I,\ R. \tag{5.17}$$

Съгласно (5.17)

$$\begin{aligned}
&p(s) = 0,\\
&p(a) = p(s) + a(s,a) = 4,\\
&p(b) = p(a) + a(a,b) = a(s,a) + a(a,b) = 4 + 2,\\
&p(c) = p(b) - a(c,b) = a(s,a) + a(a,b) - a(c,b) = 6 - 1,\\
&p(t) = p(c) + a(c,t) = a(s,a) + a(a,b) - a(c,b) + a(c,t) = p = 8.
\end{aligned}$$

Получихме



Всяка допълнителна единица на потока, протичаща по тази верига, "ще струва" 8 (лева, $ и т.н.).

В общия случай, разглежданият алгоритъм, използвайки (5.17), намира увеличаваща верига, в която сумата от цените на правите дъги минус сумата от цените на обратните дъги е равна на p (вж. последното от горните равенства).

С други думи, ако алгоритъмът намери увеличаваща верига от дъги, удовлетворяващи (5.17), нарастването на стойността за всяка нова единица от потока, пренасяна от s до t, ще бъде p.

Ще дадем формално описание на алгоритъма.

ОПИСАНИЕ НА АЛГОРИТЪМА ЗА ТЪРСЕНЕ НА ПОТОК С МИНИМАЛНА СТОЙНОСТ. *СТЪПКА 1.* За всяка дъга (x,y), $f(x,y) = 0$. Положете $p(x) = 0$, за всеки връх x.

СТЪПКА 2. (*Определяне на дъгите, в които се допуска изменение на потока:*) Определете дъгите, за които

$$p(y) - p(x) = a(x,y) \quad \text{и} \quad c(x,y) - f(x,y) > 0$$

и означете множеството от тези дъги с I. Формирайте множество R от дъги, за които

$$p(y) - p(x) = a(x,y) \quad \text{и} \quad f(x,y) > 0.$$

(Останалите дъги формират множество N).

СТЪПКА 3. (*Увеличение на потока*): Приложете алгоритъма за търсене на максимален поток (увеличаваща верига) в изходната мрежа, при намерените разпределения на дъгите в множествата I, R и N. Изпълнението на този алгоритъм приключва, когато:

а) от s до t са транспортирани дадените v единици поток или

б) полученият за така формираните множества I, R и N поток е максимален.

Ако е налице а) — край на алгоритъма за търсене на поток с минимална стойност. Полученият поток от v единици, пренесени от s до t, е поток с минимална стойност.

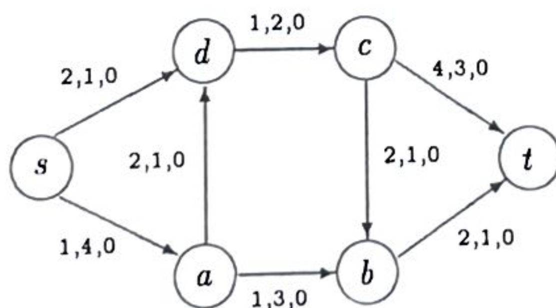
В случая б) — проверете дали полученият поток е максимален в изходния граф (вж. теореми 5.1 — 5.3 за максимален поток и минимален разрез, и алгоритъма за търсене на увеличаваща верига). Ако това е така — край! Този максимален поток (за изходния граф) има минимална стойност. В противен случай, т.е. когато намереният максимален поток при текущото разпределение на дъгите в множествата I, R, N не е максимален за изходния граф* (възможно е, когато разрезът не е наситен), преминете към *стъпка 4.*

СТЪПКА 4. (*Промяна на маркиращите числа* $p(x)$:) Изходна информация при изпълнение на тази стъпка са резултатите от процедурата за оцветяване на върховете в алгоритъма за търсене на увеличаваща верига (който е подалгоритъм на алгоритъма за търсене на максимален поток).

За всички неоцветени върхове x увеличете с 1 стойностите на маркиращите числа $p(x)$. Преминете към *стъпка 2.*

Ще илюстрираме алгоритъма за търсене на поток с минимална стойност.

ПРИМЕР 5.9. В мрежата, изобразена по-долу, на всяка дъга са съпоставени по три числа — първото е капацитетът на дъгата, второто е цената за пренос на единица продукция по дъгата, а третото е величината на потока (нулев начален поток).



*Алгоритъмът за търсене на поток с минимална стойност се изпълнява за подграф на изходния граф поради допълнителното ограничение (5.17). Ето защо полученият поток (съвкупност от дъги) може да не образува разрез в изходния граф.

В началото всички маркиращи числа са нули, т.е.

$$p(s) = p(a) = p(b) = p(c) = p(d) = p(t) = 0.$$

Всички дъги и върхове, с изключение на върха s (той ще бъде оцветен винаги) са неоцветени. За краткост ще изложим резултатите от изпълнението на алгоритъма в таблица.

Итерация	$p(s)$	$p(a)$	$p(b)$	$p(c)$	$p(d)$	$p(t)$	Оцветени дъги	Оцветени върхове
0	0	0	0	0	0	0	Няма	s
1	0	1	1	1	1	1	(s,d)	s,d
2	0	2	2	2	1	2	(s,d)	s,d
3	0	3	3	3	1	3	$(s,d),(d,c)$	s,d,c
4	0	4	4	3	1	4	$(s,d),(d,c),(s,a),(c,b)$	s,d,c,a,b
5	0	4	4	3	1	5	$(s,d),(d,c),(s,a),(c,b),(b,t)$	s,d,c,a,b,t

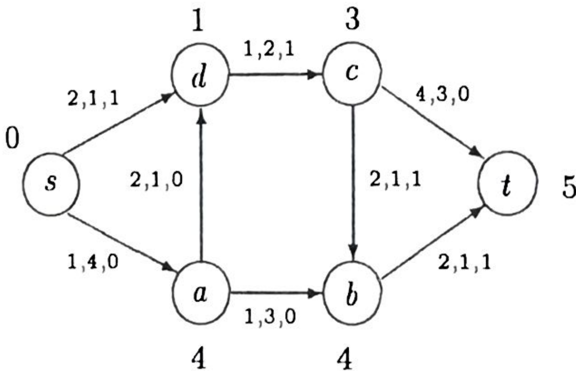
Върхът t се оказва оцветен. По увеличаващата верига

$$(s,d),\ (d,c),\ (c,b),\ (b,t),$$

можем да пропуснем една единица поток от s до t (капацитетите на дъгите от веригата не дават възможност за повече единици — дъгата (d,c) е тясно място). Увеличаваме потока в съответните дъги

$$f(s,d) = f(d,c) = f(c,b) = f(b,t) = 1.$$

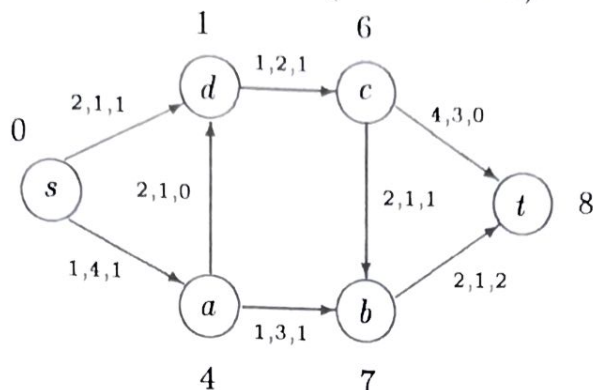
Този пренос се осъществява на цена (разходи) $p(t) = 5$. Състоянието на мрежата е:



Полученият поток не е максимален за изходния граф, защото след процедурата "оцветяване на върховете" в алгоритъма за търсене на увеличаваща верига, върхът t ще се окаже оцветен, т.е. съществува увеличаваща потока верига. Продължаваме изпълнението на алгоритъма.

Итерация	$p(s)$	$p(a)$	$p(b)$	$p(c)$	$p(d)$	$p(t)$	Оцветени дъги	Оцветени върхове
6	0	4	4	3	1	5	$(s,d),(s,a)$	s,d,a
7	0	4	5	4	1	6	$(s,d),(s,a)$	s,d,a
8	0	4	6	5	1	7	$(s,d),(s,a)$	s,d,a
9	0	4	7	6	1	8	$(s,d),(s,a),(a,b),(b,t)$	s,d,a,b,t

Върхът t се оказва оцветен. По веригата (s,a), (a,b), (b,t) можем да изпратим само една единица поток от s в t, при разходи за този пренос $p(t) = 8$. В резултат получаваме следната мрежа (с нов поток).



Ще покажем, че полученият поток е максимален за изходния граф. Процедурата "оцветяване на върховете" в алгоритъма за търсене на увеличаваща верига разбива множеството от върховете на графа на следните 2 множества.

$T = \{s, d\}$ — множество от оцветени върхове.

$\overline{T} = \{a, b, c, t\}$ — множество от неоцветени върхове.

За дъгите от разреза $\langle T, \overline{T} \rangle$, ориентирани от T към $\overline{T}$ (дъги с начало в T и край $\overline{T}$), имаме

$$(s,a) - f - \text{наситена}; \quad (d,c) - f - \text{наситена},$$

а за дъгите, ориентирани от $\overline{T}$ към T (т.е. с начало в $\overline{T}$ и край в T), имаме

$$(a,d) - f - \text{нулева}.$$

Следователно $\langle T, \overline{T} \rangle$ е минимален разрез, а f е максимален поток (вж. теореми 5.1 — 5.3).

Полученият поток f, с величина $val\,(f) = 2$ е максимален поток, с минимална стойност. Едната единица поток "протича" по веригата (s,d), (d,c), (c,b), (b,t) и разходите за това са:

$$a(s,d) + a(d,c) + a(c,b) + a(b,t) = 1 + 2 + 1 + 1 = 5 \text{ (лв., \$, } \mathcal{L} \text{ и т.н.)}$$

Другата единица на потока достига от s до t по веригата (s,a), (a,b), (b,t) и разходите за това са

$$a(s,a) + a(a,b) + a(b,t) = 4 + 3 + 1 = 8.$$

Общата стойност на потока е $5 + 8 = 13$.

ОБОСНОВКА НА АЛГОРИТЪМА [1]. Доказателството, че даденият алгоритъм наистина намира поток от v единици, чиито пренос от s до t е минимален, ще направим с помощта на теоремите за двойственост (вж. параграф 2.1).

Нека:

$p(x)$ е дуалната променлива, съответна на ограничението (5.14);

$p(s)$ е дуалната променлива, съответна на ограничението (5.12);

$p(t)$ е дуалната променлива, съответна на ограничението (5.13);

$g(x, y)$ е дуалната променлива, съответна на ограничение (5.15);

v разглеждаме като променлива (не като константа).

По-нататък ще покажем, че дуалните променливи $p(x)$ съвпадат с маркиращите числа $p(x)$, изчислявани при изпълнението на алгоритъма.

Дуалната (двойнствената) задача на задачата за търсене на минимален поток (5.12) — (5.16) е:

(5.18) $$\sum_{(x,y)} c(x, y).g(x, y) \to \min$$

при ограничителни условия

(5.19) $$-p(s) + p(t) = p,$$

(5.20) $$p(x) - p(y) + g(x, y) \geq -a(x, y), \text{ за } \forall(x, y),$$

(5.21) $$g(x, y) \geq 0, \text{ за } \forall(x, y),$$

(5.22) $$p(x) - \text{ свободна променлива (за } \forall x).$$

В дуалната задача (5.18) — (5.22), дуалното ограничение (5.19) съответства на свободната променлива v от правата за-

дача. Всяко неравенство (5.20) е дуалното ограничение, съответстващо на променливата $f(x, y)$ от правата задача.

От теоремите за дуалност (вж. параграф 2.1) следва:

$$p(x) - p(y) + g(x, y) > -a(x, y) \Rightarrow f(x, y) = 0; \tag{5.23}$$

$$g(x, y) > 0 \Rightarrow f(x, y) = c(x, y). \tag{5.24}$$

Ако положим

$$p(s) = 0,\ p(t) = p \text{ и} \tag{5.25}$$

$$g(x, y) = \max\{0,\ p(y) - p(x) - a(x, y)\},\ \text{за } \forall(x, y) \tag{5.26}$$

и вземем в предвид (5.23), очевидно следва $g(x, y) = 0$, т.е. (5.23) може да се запише във вида

$$p(y) - p(x) < a(x, y) \Rightarrow f(x, y) = 0 \tag{5.27}$$

и освен това (5.24) приема вида

$$p(y) - p(x) > a(x, y) \Rightarrow f(x, y) = c(x, y). \tag{5.28}$$

Следователно, достатъчно е алгоритъмът да намира такива стойности на $p(x)$ за върховете x и такива $f(x, y)$ за дъгите (x, y), че да бъдат изпълнени (5.25), (5.27) и (5.28). Освен това $f(x, y)$ трябва да удовлетворява условията (5.12) — (5.16) (за поток).

	1. Измененията на потока, които алгоритъмът за намиране на поток с минимална стойност осъществява, не нарушават условията (5.27), (5.28) (условието (5.25) винаги е изпълнено).

Това наистина е така. В началния момент за всеки връх x, $p(x) = 0$ ($p = 0 = p(t)$) и потокът е нулев, т.е. условията (5.27) и (5.28) са налице. Нещо повече, при всяка итерация на алгоритъма тези условия се удовлетворяват. Съвкупността от значения $f(x, y)$ винаги е поток (удовлетворени са (5.12) — (5.15)) и освен това в алгоритъма промяна на потока се допуска само в тези дъги (x, y), за които

$$p(y) - p(x) = a(x, y)$$

(вж. пример 5.9).

	2. Промяната на маркиращите числа в алгоритъма за намиране на поток с минимална стойност също не води до нарушаване на условията (5.27) — (5.28) (и (5.25), което е очевидно).

Това наистина е така. В алгоритъма маркиращите числа $p(x)$ се увеличават с единица за неоцветените върхове. Невъзможността да се оцвети един връх означава, че от източника до него не е възможен никакъв препос на единици от потока.

Когато и двата края на дъгата (x, y) са или не са оцветени, разликата $p(y) - p(x)$ не се променя.

В случая, когато върхът x е оцветен, а върхът y неоцветен, е изпълнена някоя от следните три релации:

а) $p(y) - p(x) < a(x, y)$;

б) $p(y) - p(x) > a(x, y)$;

в) $p(y) - p(x) = a(x, y)$ и $f(x, y) = c(x, y)$.

В случая а) след $p(y) := p(y) + 1$, е изпълнено

$$p(y) - p(x) \leq a(x, y),$$

т.е. не се нарушава (5.27).

В случая б) след увеличението $p(y) := p(y) + 1$, е изпълнено

$$p(y) - p(x) > a(x, y),$$

т.е. не се нарушава (5.28).

Аналогично в случая в) след увеличаване на $p(y)$ с 1 е изпълнено $p(y) - p(x) > a(x, y)$, т.е. е изпълнено (5.28).

Накрая, в случая когато върхът y е оцветен, а върхът x не е, са налице следните възможности:

а) $p(y) - p(x) < a(x, y)$;

б) $p(y) - p(x) > a(x, y)$;

в) $p(y) - p(x) = a(x, y)$ и $f(x, y) = 0$.

Ако е налице а), увеличението на $p(x)$ с 1 съхранява неравенството

$$p(y) - p(x) < a(x, y),$$

т.е. изпълнено е (5.27).

В случая б), увеличението на $p(x)$ с 1 води до

$$p(y) - p(x) \geq a(x, y),$$

т.е. не се нарушава (5.28).

Ако е налице в), увеличението на $p(x)$ с 1 води до

$$p(y) - p(x) < a(x, y),$$

т.е. изпълнено е (5.27).

От казаното следва, че алгоритъмът за намиране на поток с минимална стойност формира допустим поток, който при $p = p(t)$ удовлетворява условията (5.27) и (5.28). Ще обърнем внимание, че последното значение на $p(t)$ (което показва цената за пренос на последната единица от потока между s и t) е достатъчно голямо. По този начин оптималността на намерения поток, определена с (5.16), гарантира оптималност на потока, определен с (5.11).

Алгоритъмът за намиране на минимален поток завършва след краен брой стъпки. По същество алгоритъмът свършва след пренос на последната единица от потока. Да припомним, че в началото наложихме изискването $a(x, y)$, т.е. цените на дъгите да бъдат крайни, цели положителни числа. Тогава очевидно и $p(t)$ ще бъде положително цяло число. Тъй като алгоритъмът за минимален поток изпълнява не повече от $p(t) + 1$ пъти алгоритъма за търсене на максимален поток, а последният е с краен брой стъпки (Едмондс и Карп), то следва, че алгоритъмът за търсене на поток с минимална стойност също е с краен брой стъпки.

Наложеното до този момент ограничение за целочисленост на цените $a(x, y)$ даваше възможност да разглеждаме целочислени стойности p и всички маркиращи числа $p(x)$ да увеличаваме с единица. Ясно е, че стойността за пренос на единица поток от s до t може да не е цяло положително число. Това налага в алгоритъма да се разглеждат и нецелочислени стойности за параметъра p. С други думи трябва да знаем какви стойности да приписваме на параметъра p и какви нараствания ще имат маркиращите числа. Ще покажем как трябва да се модифицира алгоритъмът, за да работи при произволни положителни цени $a(x, y)$.

Нека алгоритъмът е изпълнен за някое $p = p(t)$. Възможни са следните три ситуации.

ВАРИАНТ 1: Нека в дъгата (x, y) върхът x е оцветен, а върхът y не е оцветен. Дъгата (x, y) ще бъде оцветена в следващата итерация само когато $(x, y) \in I$ (множеството на увеличаващите дъги) и изменението на $p(y)$ е такова, че е налице

$$p(y) - p(x) = a(x, y).$$

Тъй като (5.28) не трябва да се нарушава, при $(x, y) \in I$, то

$$p(y) - p(x) \leq a(x, y).$$

Следователно, за да бъде оцветена дъгата (x, y) при следващата итерация, $p(y)$ трябва да се увеличи с $\delta(x, y)$, където

$$\delta(x, y) = a(x, y) + p(x) - p(y). \tag{5.29}$$

ВАРИАНТ 2: Нека сега върхът y е оцветен, а върхът x не е. В този случай дъгата (x, y) може да бъде оцветена в следващата итерация само ако $(x, y) \in R$ (множеството от намаляващи дъги) и изменението на $p(x)$ е такова, че

$$p(y) - p(x) = a(x, y).$$

Поради (5.27), ако $(x, y) \in R$, то

$$p(y) - p(x) \geq a(x, y).$$

Следователно, за да бъде оцветена дъгата (x, y) в следващата итерация, $p(x)$ трябва да се увеличава с $\delta(x, y)$, където

$$\delta(x, y) = p(y) - p(x) - a(x, y). \tag{5.30}$$

ВАРИАНТ 3: Нека сега и двата върха x и y на дъгата (x, y) са оцветени или неоцветени. В този случай се полага

$$\delta(x, y) = \infty. \tag{5.31}$$

Предвид (5.29) — (5.31), да определим величината Δ по следния начин:

$$\Delta = \min_{(x,y)}\{\delta(x, y)\} > 0. \tag{5.32}$$

Ако дуалната променлива $p(x)$ за всеки неоцветен връх x се увеличи с Δ, то в следващата итерация ще бъде оцветена поне една нова дъга. Такова изменение на дуалните променливи ще води до увеличение на $p(t)$ с Δ. По аналогичен начин се определят следващите нараствания на двойнствените променливи, като винаги се гарантира оцветяването на поне една нова дъга. Разбира се, когато се стигне до $\Delta = \infty$, е невъзможно оцветяването на нови дъги — текущият поток е максимален. Така модифицираният алгоритъм за намиране на поток с минимална стойност увеличава вероятността за построяване в мрежата на нови увеличаващи вериги. Освен това тази модификация на алгоритъма решава проблема след краен брой стъпки. Тъй като броят на различните вериги от s до t е краен, а $p(t)$ дава винаги сумарната цена на дъгите от някоя $(s - t)$ верига, то $p(t)$ ще приема краен брой различни значения — следователно и алгоритъмът ще приключи след краен брой стъпки.

6. Алгоритъм на дефекта

Ще разгледаме сега един алгоритъм, предложен от Форд и Фалкерсон [10], за намиране на поток с минимална стойност — *алгоритъм на дефекта.* При този алгоритъм отрицателни значения за цените $a(x, y)$ ще бъдат допустими (за разлика от досегашното изискване за положителност на тези величини). Ограничението, което ще бъде наложено при използване алгоритъма на дефекта, е в изходния граф да не съществува цикъл с неограничен капацитет и отрицателна сумарна цена. Липсата на такова ограничение прави задачата несъдържателна. Неограничен брой минавания на единици от потока по такъв цикъл ще води до неограничено намаляване стойността на преминаващия поток.

В много практически задачи на потоците в дъгите освен ограничението отгоре — $f(x, y) \leq c(x, y)$, се налага и ограничение отдолу — $0 \leq l(x, y) \leq f(x, y)$, за $\forall(x, y)$.

Освен това за разлика от изложения вече алгоритъм за намиране на поток с минимална стойност в този алгоритъм, началният поток не е задължително да бъде нулев. В алгоритъма неотрицателното число $l(x, y)$, споменато по-горе, се нарича *минимална пропускателна способност (минимален капацитет)*, а досега разглежданата пропускателна способност $c(x, y)$ се нарича *максимална пропускателна способност (максимален капацитет)*. Вече разгледаният алгоритъм за намиране на поток с минимална стойност е частен случай на алгоритъма на дефекта

при $l(x, y) = 0$, за $\forall(x, y)$.

ИДЕЯ НА АЛГОРИТЪМА [1]. Да добавим към изходния граф една "възвратна дъга" (t, s), по която целият поток, който се изпраща от s в t, обратно се връща в източника s. В така *разширения граф* очевидно чистият поток във всеки връх ще бъде нула. На много места в литературата такива потоци се наричат *циркулации*.

Всеки поток в изходния граф е еквивалентен на някоя циркулация и обратно. За възвратната дъга (t, s) се полага (при търсене на поток с минимална стойност)

$$l(t, s) = c(t, s) = v \quad \text{и} \quad a(t, s) = 0.$$

Ако си поставим задача да търсим $(s - t)$ поток с минимална стойност, който е максимален $(\max .v)$, се прави полагането

$$l(t, s) = 0, \; c(t, s) = \infty \quad \text{и} \quad a(t, s) = -p,$$

където p е достатъчно голямо число. Например, p както преди е по-голямо от максималната цена за $(s-t)$ - преноса на единица от потока по мрежата.

При така направените бележки (въведена възвратна дъга), можем да формулираме задачата за намиране на поток с минимална стойност, при ненулеви минимални капацитети на дъгите, като задача на линейното оптимиране. Ще формулираме и нейната дуална задача.

Права задача

(5.33) $$\sum_{(x,y)} a(x, y) f(x, y) \to \min$$

при ограничения за $\forall x$

(5.34) $$\sum_{y} f(x, y) - \sum_{y} f(y, x) = 0,$$

(5.35) $$l(x, y) \leq f(x, y) \leq c(x, y), \quad \text{за} \quad \forall(x, y).$$

Дуална задача

(5.36) $$\sum_{(x,y)} [-c(x,y)g_2(x,y) + l(x,y)g_1(x,y)] \to \max$$

при ограничения за $\forall(x,y)$

(5.37) $$p(y) - p(x) + g_1(x,y) - g_2(x,y) \leq +a(x,y)$$

(5.38) $$g_1(x,y) \geq 0$$

(5.39) $$g_2(x,y) \geq 0$$

Дуалната променлива, съответна на (5.34) за върха x, сме обозначили с $p(x)$, а дуалните променливи $g_1(x,y)$ и $g_2(x,y)$ са дуални, съответно на лявото и дясното ограничение в (5.35). Да направим сега следните полагания и предположения:

(5.40) $g_1(x,y) - g_2(x,y)$ означаваме с $g(x,y)$ — променлива с произволен знак;

(5.41) $g(x,y) = a(x,y) + p(x) - p(y)$, за всяка дъга (x,y);

(5.42) $g_1(x,y) = g(x,y)$ и $g_2(x,y) = 0$, при $g(x,y) \geq 0$;

(5.43) $g_1(x,y) = 0$ и $g_2(x,y) = -g(x,y)$, при $g(x,y) < 0$.

От условията за спрегнатост на правата и дуалната задача (вж. параграф 2.1) ще получим:

(5.44) $$\begin{aligned} g_1 > 0 &\Rightarrow f(x,y) = l(x,y), \\ g_2 > 0 &\Rightarrow f(x,y) = c(x,y). \end{aligned}$$

Поради (5.41), условията (5.44) могат да се запишат така:

(5.45) $$p(y) - p(x) < a(x,y) \Rightarrow f(x,y) = l(x,y),$$

(5.46) $$p(y) - p(x) > a(x,y) \Rightarrow f(x,y) = c(x,y).$$

(Получените условия са твърде сходни с условията (5.27) и (5.28)). За краткост, ако положим

(5.47) $$\alpha(x,y) = a(x,y) + p(x) - p(y), \text{ за всяка дъга } (x,y),$$

получените условия могат да се запишат като

(5.48) $$\alpha(x,y) > 0 \Rightarrow f(x,y) = l(x,y),$$

(5.49) $$\alpha(x,y) < 0 \Rightarrow f(x,y) = c(x,y).$$

От казаното следва, че за да решим правата задача, трябва да намерим поток $f(x,y)$, удовлетворяващ (5.34) и маркиращи числа $p(x)$, удовлетворяващи (5.48) и (5.49). (При това, условията (5.37) — (5.39) автоматично ще се изпълняват.)

Да допуснем, че такъв поток е намерен, т.е. $f(x,y)$ са такива, че чистият поток през всеки връх е нула и да допуснем, че сме приписали някакви значения на маркировките $p(x)$. Тогава всяка дъга на изходния граф може да се намира в някое от следните девет различни състояния:

Състояние	Величина на дефекта
(1). $\alpha(x,y) < 0$ $f(x,y) < c(x,y)$	$\alpha(x,y)[f(x,y) - c(x,y)]$
(2). $\alpha(x,y) < 0$ $f(x,y) = c(x,y)$	0
(3). $\alpha(x,y) < 0$ $f(x,y) > c(x,y)$	$f(x,y) - c(x,y)$
(4). $\alpha(x,y) = 0$ $f(x,y) < l(x,y)$	$l(x,y) - f(x,y)$
(5). $\alpha(x,y) = 0$ $l(x,y) \leq f(x,y) \leq c(x,y)$	0
(6). $\alpha(x,y) = 0$ $f(x,y) > c(x,y)$	$f(x,y) - c(x,y)$
(7). $\alpha(x,y) > 0$ $f(x,y) < l(x,y)$	$l(x,y) - f(x,y)$
(8). $\alpha(x,y) > 0$ $f(x,y) = l(x,y)$	0
(9). $\alpha(x,y) > 0$ $f(x,y) > l(x,y)$	$\alpha(x,y)[f(x,y) - l(x,y)]$

Таблица 5.1

Освен споменатите девет състояния, в таблицата са дадени и съответните, т.нар. *величини на дефекта*. Те характеризират дефектността на дъгата — степента на неизпълнение за тази дъга на условията от теорема 9.2, т.е. (5.48) и (5.49).

С $k(x, y)$ ще бележим *величината на дефекта* на дъгата (x, y). Проверете, че за всяка от горните девет ситуации винаги е изпълнено $k(x, y) \geq 0$. Ще означаваме с k сумата от дефектите на всички дъги на графа. Обърнете внимание и на това, че дъгите, за които условията (5.48) и (5.49) са изпълнени, имат дефект нула. В останалите случаи стойността на дефекта е положителна.

ИДЕЯ НА АЛГОРИТЪМА НА ДЕФЕКТА. Същината на алгоритъма на дефекта е в последователното намаляване до нула на дефекта на някоя дъга, без това да увеличава дефекта на останалите дъги. По този начин дъгите на изходния граф ще изпълняват условията (5.48), (5.49) и алгоритъмът ще формира максимален поток с минимална стойност. Намаляването до нула на дефекта $k(x, y)$ се извършва чрез намаляване или увеличаване на потока $f(x, y)$.

СЛУЧАЙ 1: Нека за дъгата (x, y), $k(x, y) > 0$, т.е. дъгата (x, y) е дефектна и потокът в дъгата (x, y) трябва да се увеличи, за да бъде отстранен дефектът. В този случай се търси верига от върха y във върха x, по която потокът може да се увеличи, без това увеличение да влияе негативно върху дефектите на останалите дъги на графа. Когато такава верига бъде намерена, тя очевидно ще образува заедно с дъгата (x, y) контур, по който може да се осъществи допълнителен пренос на поток, увеличаващ потока по дъгата (x, y). При това не нараства дефектът на нито една дъга. Тези увеличения на потока се повтарят, докато дъгата (x, y) стане недефектна, т.е. $k(x, y) = 0$, или е невъзможно намирането на увеличаваща потока верига. Ако е налице последното, т.е. липсва увеличаваща верига, "алгоритъмът увеличава" маркиращите числа на някои от върховете, след което отново се търсят увеличаващи потока вериги от y в x. Увеличаването на маркиращите числа се прави така, че дефектът на никоя дъга не нараства. Крайните възможни изходи са два: или дъгата (x, y) става недефектна, или се оказва, че в мрежата не съществува поток, за който

$$l(x, y) \leq f(x, y) \leq c(x, y).$$

СЛУЧАЙ 2: Нега сега за отстраняването на дефекта $k(x, y) > 0$ се налага потокът да бъде намален. Повтарят се действията, извършени в горния *СЛУЧАЙ 1*, с тази разлика, че се строи увеличаваща потока верига от върха x във върха y (а не от y в x). Ще дадем описание на алгоритъма.

ОПИСАНИЕ НА АЛГОРИТЪМА НА ДЕФЕКТА [1]. СТЪПКА *1.* Изберете напълно произволен поток, за който сумарният поток във всеки връх от графа е нула, т.е. $f(x,y)$ удовлетворява (5.34), без да е задължително да удовлетворява (5.35). Изберете и напълно произволен набор от стойности за маркиращите числа $p(x)$ на върховете x.

СТЪПКА 2. (*Изчисляване на дефектите*) За всяка дъга (x,y) изчислете $\alpha(x,y)$ и $k(x,y)$, използувайки (5.47) и табл. 5.1. Ако дефектите на всички дъги са нула — край на алгоритъма. Полученият поток $f(x,y)$ е поток с минимална стойност. В противен случай преминете към *стъпка 3.*

СТЪПКА 3. (*Класификация на дъгите*) Формирайте множество от увеличаващи дъги I и множество от намаляващи дъги R. За всяка дъга (x,y) ако:

а) $\alpha(x,y) \geq 0$ и $f(x,y) < l(x,y)$ или

б) $\alpha(x,y) \leq 0$ и $f(x,y) < c(x,y)$,

то дъгата $(x,y) \in I$. За тази увеличаваща дъга (x,y), положете

$$i(x,y) = l(x,y) - f(x,y), \quad \text{при } \alpha(x,y) > 0,$$

$$i(x,y) = c(x,y) - f(x,y), \quad \text{при } \alpha(x,y) \leq 0.$$

а) $\alpha(x,y) \geq 0$ и $f(x,y) > l(x,y)$ или

б) $\alpha(x,y) \leq 0$ и $f(x,y) > c(x,y)$,

то дъгата $(x,y) \in R$. За тази намаляваща дъга (x,y), положете

$$r(x,y) = f(x,y) - l(x,y), \quad \text{при } \alpha(x,y) \geq 0,$$

$$r(x,y) = f(x,y) - c(x,y), \quad \text{при } \alpha(x,y) < 0.$$

Изберете произволна дефектна дъга (x,y), т.е. за която $k(x,y) > 0$. Ако $(x,y) \in I$, считайте върха y за s, а върха x за t и преминете към *стъпка 4.* Ако $(x,y) \in R$, считайте върха x за s, а върха y за t и преминете към *стъпка 4.*

(Няма дъги, едновременно принадлежащи на множеството I и R, поради $k(x,y) > 0$.)

СТЪПКА 4. (*Прилагане на алгоритъма за търсене на максимален поток*) За формираните в предишната стъпка множества I

и R и за получените пак там $i(x,y)$ и $r(x,y)$, приложете алгоритъма за търсене на максимален поток от s в t. Получения поток балансирайте с потока по дъгата, съединяваща върховете s и t. Продължете итерационната процедура за нарастване на потока, докато дъгата, съединяваща върховете s и t, стане недефектна или до получаването на максимален поток. В първия случай преминете към *стъпка 2*, а във втория преминете към *стъпка 5*.

СТЪПКА 5. (*Увеличение на маркиращите числа*) Преход към тази стъпка се извършва, когато в алгоритъма за търсене на максимален поток не могат да се построяват повече увеличаващи потока вериги. Нека S е множеството от оцветените върхове на последната итерация на алгоритъма при търсене на увеличаваща потока верига. Да означим с $\overline{S}$ неоцветените върхове при тази итерация.

Ясно е, че $s \in S$ и $t \in \overline{S}$.

Определете множествата от дъги E_1 и E_2 по следния начин:

$$E_1 = \{(x,y) | x \in S,\ y \in \overline{S},\ \alpha(x,y) > 0,\ f(x,y) \le c(x,y)\} \tag{5.50}$$

$$E_2 = \{(y,x) | x \in S,\ y \in \overline{S},\ \alpha(y,x) < 0,\ f(y,x) \ge l(y,x)\} \tag{5.51}$$

Ако $E_1 = \emptyset$, то положете $\Delta_1 = \infty$.

В противен случай, положете

$$\Delta_1 = \min_{E_1}\{\alpha(x,y)\} > 0. \tag{5.52}$$

Ако $E_2 = \emptyset$, то положете $\Delta_2 = \infty$.

В противен случай, положете

$$\Delta_2 = \min_{E_2}\{\alpha(x,y)\} > 0. \tag{5.53}$$

Накрая, положете

$$\Delta = \min\{\Delta_1, \Delta_2\} > 0. \tag{5.54}$$

Ако $\Delta = \infty$ — край. В изходния граф не съществува допустим поток.

Ако $\Delta < \infty$, за всяко $x \in \overline{S}$ увеличете маркиращото число $p(x)$ с Δ, т.е. $p(x) := p(x) + \Delta$.

Върнете се към *стъпка 3*.

Обосновката на изложения алгоритъм на дефекта можете да намерите в [1], [10] и др. Основен недостатък на този алгоритъм се явява произволният избор на началните стойности на маркиращите числе $p(x)$. В повечето случаи няма аргументи за това, как трябва да се осъществи този избор — това понякога води до големи стойности за дефекта на някои дъги, а оттам и голям брой нараствания на потока за отстраняване на дефекта.

7. Динамични потоци в мрежа

Досега разглежданите потоци бяха статични. Дъгите в изходния граф имаха пропускателни способности и цени. В много задачи се търсят потоци в мрежа такива, че всяка единица да преминава от източника s в стока t, за време, непревишаващо дадена стойност. На всяка дъга (x, y) в тези случаи се приписва цяло положително число $a(x, y)$, наречено *време за пренос* през дъгата (количеството времеви интервали, необходими за пренос на единица поток).

Максималното количество единици на потока, които могат да "влязат" в дъгата (x, y) в момента $T = 0,\ 1,\ 2, \ldots$, се обозначава със $c(x, y, T)$, т.е. времето е дискретно и T е номер на времеви интервал.

	Динамичен поток се нарича поток от s до t, при който във всеки момент от време T, във всяка дъга (x, y) не влизат повече от $c(x, y, T)$ единици поток.

	Максимален динамичен поток от s до t за период от p времеви интервала, е такъв динамичен поток, при който от s до t за период от време p, се пренасят максимален възможен брой единици поток.

ПРИМЕР 5.10. Транспортна фирма трябва да препрати в рамките на 18 часа 135 пътника от Благоевград в Пловдив. Това очевидно се свежда до задача за намиране на максимален динамичен поток. Нека Благоевград е източникът s, а Пловдив е стокът t. Нека всеки град, принадлежащ на възможен маршрут (Велинград, Ихтиман, София, Боровец и т.н.), разглеждаме като връх на граф, а възможните рейсове на фирмата между населените места — като дъги (ребра) на този граф. Нека времето за превоз по всяка дъга

(x, y) е времето на пътуване между градовете x и y, закръглено до часове (в това време трябва да се включи и времето за престой при смяна на рейса). Нека $c(x, y, T)$, т.е. пропускателната способност на дъгата (x, y) в момента T, е броят на местата в съответния рейс с отправно време T. Когато такъв рейс няма, се полага $c(x, y, T) = 0$.

Разглежданата задача има решение, ако в описания по-горе граф съществува динамичен поток от 135 единици между Благоевград и Пловдив за период от 18 интервала от време.

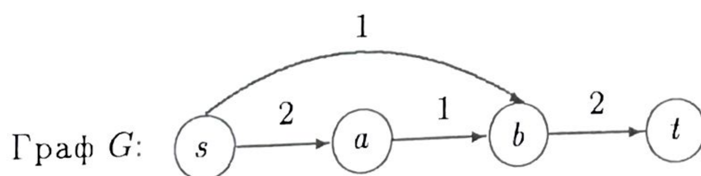
Очевидно намирането на максимален динамичен поток е по-сложно от намирането на максимален поток. Налага се проследяването на движението на всяка единица от потока, с цел във всеки момент от време на входа на всяка дъга да не се нарушава капацитетът ѝ. Ще сведем задачата за максимален динамичен поток към задачата за максимален поток.

Ако разгледаме *разгърнат във времето вариант на изходния граф* $G = (V, E)$, определен така:

$G_p = (V_p, E_p)$ – разгърнат граф, където
$V_p = \{x_i | x \in V,\ i = 0,\ 1, ...,\ p\}$,
$E_p = \{(x_i, y_j) | (x, y) \in E,\ i = 0,\ 1, ...,\ p - a(x, y),\ j = i + a(x, y)\}$,
$c(x_i, y_j) = c(x, y, i)$.

Очевидно множеството върхове V_p се получава от V след p-кратно дублиране на върховете и в графа G_p върховете x_i и y_j се съединяват с дъга (x_i, y_j), ако в G потокът протича от x до y за време $j - i$. Освен това, всеки динамичен $(s - t)$ поток в графа G е еквивалентен на поток между множество източници и стокове в G_p и обратно.

ПРИМЕР 5.11. Нека в графа G



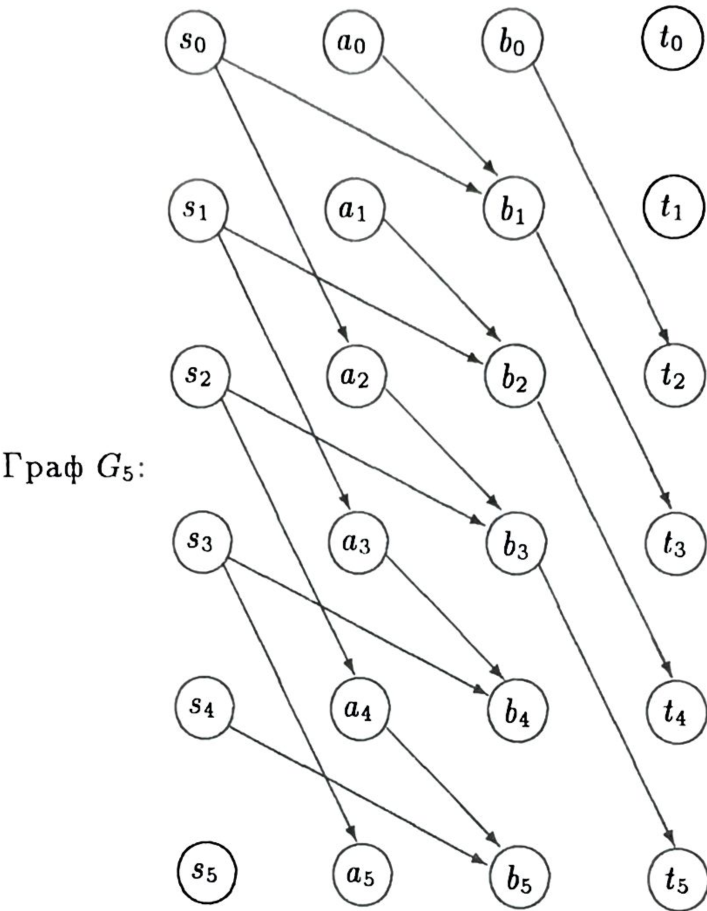
пропускателните способности на всички дъги са 1, а числата, дадени над всяка дъга в графа, са времената за пренос по съответната дъга.

Да разгледаме следния динамичен поток:

Път	Отправен момент T	Количество
$s,\ a,\ b,\ t$	0	1 единица
$s,\ b,\ t$	1	1 единица
$s,\ b,\ t$	2	1 единица

Не е трудно да се определи еквивалентният му статичен поток в разгърнатия по време граф ($p = 5$) G_5.

Граф G_5:



Статичен поток в разгърнатия граф	
Път	Количество
$s_0,\ a_2,\ b_3,\ t_5$	1 единица
$s_1,\ b_2,\ t_4$	1 единица
$s_2,\ b_3,\ t_5$	1 единица

И така всеки динамичен поток в графа G е еквивалентен на статичен поток в разгърнатия във времето (период p) вариант на изходния граф. Ясно е, че алгоритъмът за намиране на максимален (статичен) поток в разгърнатия вариант на графа ще върши работа и за намиране на максимален динамичен поток. Оказва се обаче, че при големи p търсенето на максимален поток в G_p изисква голям обем операции. Форд и Фалкерсон [10] са разработили ефективен алгоритъм специално за търсене на максимален динамичен поток, който ползва като подалгоритъм алгоритъма за търсене на поток с минимална стойност.

ИДЕЯ НА АЛГОРИТЪМА. Да припомним, че в алгоритъма за търсене на поток с минимална стойност се правеха последователно опити, от s до t да се пренесат максимален брой единици

от потока на цена 1, 2, 3 и т.н., докато се получи максимален поток. Ако $f_1, f_2, \ldots, f_p$ са $(s-t)$-потоците, формирани при изпълнението на алгоритъма за търсене на поток с минимална стойност, то всеки от тях може да се представи като набор от пътища, водещи от s до t, по които се пренасят някакви количества формиращи потока f_i:

$f_i: f_{i,1}, f_{i,2}, \ldots, f_{i,r_i}$ (*Разлагане на потока по пътища*)

Да означим с $n_{i,j}$ броя единици на потока, които протичат по пътя $f_{i,j}$ в потока f_i и с $a(f_{i,j})$ общата цена на пътя $f_{i,j}$. Очевидно, за всеки път $f_{i,j}$ цената му не надвишава i, тъй като пътят възниква след прилагане на алгоритъма за поток с минимална стойност.

След казаното, формалното описание на алгоритъма е следното.

ОПИСАНИЕ НА АЛГОРИТЪМА ЗА ТЪРСЕНЕ НА МАКСИМАЛЕН ДИНАМИЧЕН ПОТОК. *СТЪПКА 1.* Считайте времето за пренос по дъгите (x, y) на изходния граф за цената $a(x, y)$ от алгоритъма за търсене на поток с минимална стойоност и приложете споменатия алгоритъм, докато получите потока f_p, определен от $(s-t)$ пътищата

$$f_p: f_{p_1}, f_{p_2}, \ldots, f_{p,r_p},$$

по които протичат съответно $n_{p,1}, n_{p,2}, \ldots, n_{p,r_p}$ единици от потока.

СТЪПКА 2. За $j = 1, 2, \ldots, r_p$, пропуснете по всеки път $f_{p,j}$, $n_{p,j}$ единици от потока във всеки момент $0, 1, \ldots, p - a(f_{p,j})$. Полученият след изпълнение на тази стъпка поток е максимален динамичен поток за p единични времеви интервали.

Този алгоритъм по същество се свежда до алгоритъма за търсене на поток с минимална стойност, в който времето за пренос през дъгата се използва за тегло.

Очевидно, последният получен поток се подлага на декомпозиция — потоци по $(s-t)$ пътища. По всеки от пътищата се пропускат съответните единици поток в моментите 0, 1, 2 и т.н. до момента време, за който потокът все още успява да достига в стока t към момента p.

Обръщаме внимание, че предложеният алгоритъм се използва само за независещи от времето входни пропускателни способности.

За повече подробности, както и за *лексикографски динамични потоци* вж. [1].

2.6. Мрежово планиране и управление

Всеки голям крупен проект се реализира след изпълнение на определен брой етапи (операции). Например, построяването на една голяма сграда включва следните етапи (без претенции за прецизност и изчерпателност): маркиране и почистване на терена, изкопни работи, нулев цикъл, изпълнение на носещи конструкции (кофраж, плочи, стени и др.), изграждане покривна конструкция, довършителни работи (дограми, мазилки, ел. инсталация, В и К, и др.), вертикално планиране и озеленяване, и др.

Някои от етапите (работите, операциите) могат да се извършват едновременно, други — само последователно. Например, някои довършителни работи могат да се изпълнят едновременно с озеленяването, но не може да се полага мазилка преди операцията "зидане".

Най-общо управлението на проекта се състои в това: като се отчита времето за изпълнение на всяка операция и последователността на тяхното изпълнение, да се обезпечи своевременното завършване на обекта.

Един възможен начин за описване на подобен род проблеми е следният: можем да разгледаме мрежа, в която всеки връх на графа представлява някакъв етап, а съществуването на дъгата (x_i, x_j) показва, че етапът i задължително предшества етапа j. Теглата t_{ij}, приписани на дъгите, представляват минималното време между началото на етапа i и началото на етапа j. При тази интерпретация, очевидно теглата t_{ij} са в общия случай различни и освен това разглеждания граф е ориентиран и ацикличен. Предположението за съществуване на цикъл води до възможността за повтаряне на извършен вече етап — нещо, което не описва адекватно практиката.

В задачата най-често се търси минималното време за завършване на проекта, т.е. пътят с най-голяма дължина между върха s (начало) и върха t (край), изобразяващ завършването на всички необходими за реализацията на проекта работи. Този най-дълъг път се нарича *критичен път*, а неговите етапи определят пълното време за реализация на проекта. Всяко забавяне на изпълнението на някои от тези етапи води до забавяне на проекта като цяло.

Очевидно е сходството на разглежданата задача със задачата за намиране на най-кратък път. Нещо повече, поради очевидното неравенство $t_{ij} \geq 0$, може да бъде приложен алгоритъмът на Дийкстра за намиране на **НКП**, стига всички операции min

в този алгоритъм да се заменят с операцията max. Неотчитането обаче на специфичната структура на графа (ориентиран и ацикличен), прави алгоритъма в конкретния случай неефективен, както вече споменахме.

Възможна е и друга малко по-различна интерпретация на проблема за намиране на критичен път, която ние ще изложим.

1. Метод за намиране на критичен път

Нека сега разгледаме ориентиран граф, в който дъгите представляват някаква *операция* (етап, работа), а върховете на графа — някакви *абстрактни събития*, посочващи начало или край на етапа. При това графът е построен така, че ако една операция е представена с дъгата (x, y), то във върха x влизат само дъги, представляващи операции, *непосредствено предшестващи* дадената операция.

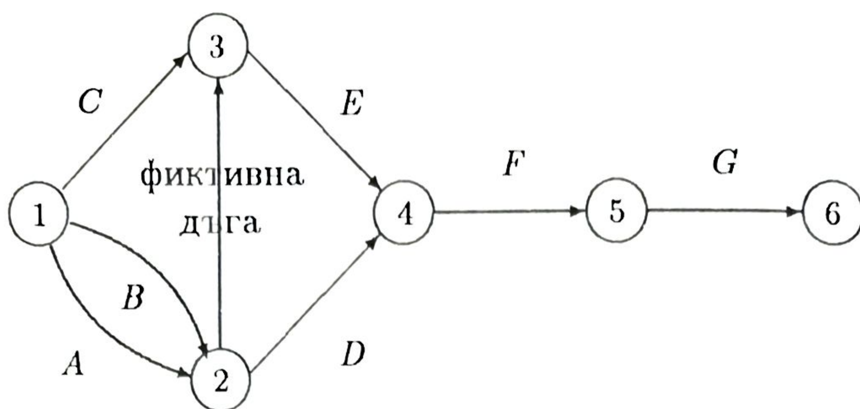
По този начин всеки проект може да се представи с описания граф, който се нарича *мрежови график*. В него са зададени всички операции на проекта, времето $t(x, y)$, необходимо за изпълнение на всяка операция и операциите, непосредствено предшестващи всяка операция.

В такова представяне обаче, релацията "предшестване" невинаги е описана точно. За отстраняване на това се добавят при необходимост *фиктивни дъги*, които представляват фиктивни операции (несъществуващи операции), с време на изпълнение $t(x, y) = 0$.

ПРИМЕР 6.1. Да разгледаме следния проект:

Операция	Време за изпълнение	Предшестващи операции
A		няма
B		няма
C		няма
D		A, B
E		A, B, C
F		D, E
G		F

Операциите, предхождащи операцията D, са подмножество на множеството операции, предхождащи операцията E. Очевидно за да осигурим предхождането на E от операциите A и B, се налага да въведем фиктивна (никаква) операция чрез фиктивна дъга (2,3), както е показано на графа по-долу.



В мрежовия график, фиктивната дъга (2,3) има тегло (време за изпълнение), $t(2,3) = 0$ (теглата на всички фиктивни дъги винаги се полагат равни на нула). В практиката паралелните дъги понякога се заменят с една дъга, представляваща съвместна операция.

И така, в графа, описващ отношението "предшестване" между етапите, т.е. в мрежовия график, дъгите представляват реални или фиктивни операции — първите, с време на изпълнение $t(x, y) > 0$, а вторите с време на изпълнение $t(x, y) = 0$. Върховете на мрежовия график наричаме *събития*. Ще казваме, че събитието x е *настъпило*, ако всички операции, изобразяващи се с дъги, влизащи във върха x, са извършени. Ясно е, че в мрежовия график не трябва да има цикли поради това, че в такъв случай проектът никога няма да бъде завършен (ние разглеждаме реални проекти с крайна реализация във времето).

Отсъствието на цикли в мрежовите графици, дава възможност да номерираме събитията (върховете) с числата 1, 2, 3,..., така че за всяка дъга (x, y) да бъде изпълнено $x < y$. Този процес се нарича *топологическа сортировка* на върховете. Как се извършва това?

2. Топологическа сортировка на върховете в ориентирани ациклични графи

Ще покажем, че във всеки ориентиран, ацикличен граф $G = (V, E)$ с n върха, можем да номерираме върховете с цели числа от множеството $\{1, 2, ..., n\}$, така че за всяка дъга $(x, y) \in E$ да бъде изпълнено $x < y$ (номерът на началния връх на дъгата е по-малък от номера на крайния връх).

▷ *ТЕОРЕМА 6.1. Във всеки ацикличен ориентиран граф има поне един връх с нулева полустепен на входа и поне един връх с нулева полустепен на изхода.*

Доказателство: Да означим с P максималния ориентиран път

$$P: \ (v_1, v_2), \ (v_2, v_3), \ (v_3, v_4), \dots, \ (v_{k-1}, v_k)$$

в графа G. Ще докажем, че за върха v_1 полустепента на входа му е нула (т.е. в него не влизат дъги) и полустепента на изхода на върха v_k е също нула (т.е. от върха v_k не излизат дъги).

Да допуснем противното, т.е. в графа G съществува дъга (u, v_1).

Ако $u \neq v_1, \ v_2, \dots, \ v_k$, то ще съществува път P_1

$$P_1: \ (u, v_1), \ (v_1, v_2), \dots, \ (v_{k-1}, v_k),$$

който съдържа всички дъги на предишния път P. Но това противоречи на условието пътя P да бъде максимален.

Ако $u \equiv v_i, \ 1 \leq i \leq k$, то в графа G ще има цикъл

$$\gamma: \ (v_1, v_2), \ (v_2, v_3), \dots, \ (v_i, v_1),$$

което противоречи на условието за ацикличност на графа.

И така, не съществува дъга (u, v_1), т.е. върха v_1 е с нулева полустепен на входа.

Аналогично се доказва, че v_k има нулева полустепен на изхода. ◁

Доказаната теорема лежи в основата на следния алгоритъм за сортиране на върховете.

ОПИСАНИЕ НА АЛГОРИТЪМА ЗА ТОПОЛОГИЧЕСКА СОРТИРОВКА

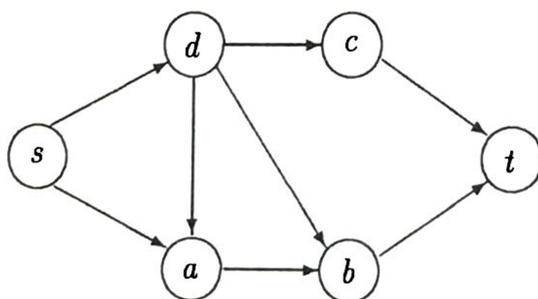
СТЪПКА 1. В графа G с n върха изберете произволен връх с нулева полустепен на изхода (съществуването му се гарантира от теорема 6.1). Номерирайте този връх с числото (номер) n. Ако $n = 1$ — край. В противен случай — преминете към *стъпка 2.*

СТЪПКА 2. Отстранете от графа този връх и инцидентните с него дъги. Означете новия граф с G и $n := n - 1$. Преминете към *стъпка 1.*

За по-голяма простота ще разглеждаме проекти, в чиито мрежови графици има точно едно събитие, което няма предшестващи (в което не влиза нито една дъга) и точно едно събитие, от което не излизат дъги. Тези събития ще наричаме съответно начално и крайно събитие, ще ги бележим със s и t, по аналогия с понятията източник и сток на мрежата. В мрежи с единствено начало s и край t, топологичната сортировка можете да извършите и така.

> *Топологична сортировка.* Присвоете на събитието s номер 1. Следващия номер присвоете на произволен неномериран връх, за който всички предшестващи събития са вече номерирани (поне едно такова събитие съществува, поради ацикличността на графа). Повторете това, докато всички събития бъдат номерирани. Очевидно крайното събитие t ще получи последния, най-голям номер n (n е броят на върховете).

ПРИМЕР 6.2. Да се номерират топологично върховете (събитията) в следния граф:



Номерирайте началното събитие s с 1. Единственото събитие, което може да се номерира след това, е събитието d. Номерирайте d с 2. След това, както събитието a, така и събитието c, могат да бъдат номерирани с 3. Номерираме върха a с 3. Номерираме върха c (или b) с номер 4. Номерираме b (или c) с 5. Накрая номерираме върха t с 6.

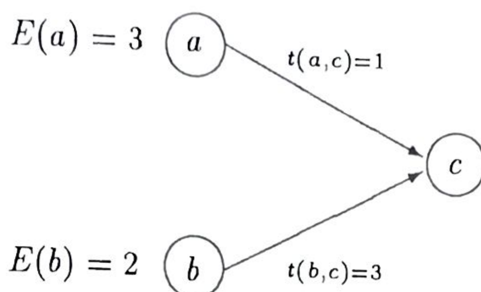
Очевидно е, че топологичната сортировка на върховете в общия случай може да се извърши по различни начини.

Нека анализираме сега мрежовия график, с цел да разберем кога най-рано или най-късно ще бъде завършен проектът, описан с него. Важно е да определим кои операции (етапи) се явяват "критични" за неговата (своевременна) крайна реализация.

Да означим с $E(x)$ *най-ранния срок* от всички възможни срокове за настъпване на събитието $x \in V$ (от *Early*).

Да означим с $L(x)$ *най-късния срок* за настъпване на събитието $x \in V$ (от *Latest* (x)), който позволява своевременното завършване на проекта.

ПРИМЕР 6.3. Да се определи най-ранният срок за настъпване на събитието c в следния граф.



Тъй като събитието c се предшества непосредствено от събитието a, то времето за нарастване на събитието c не може да бъде по-малко от

$$E(a) + t(a,c) = 3 + 1 = 4.$$

В същото време, събитието c се предхожда и от събитието b, следователно времето за настъпването на събитието c не може да бъде по-малко от

$$E(b) + t(b,c) = 2 + 3 = 5.$$

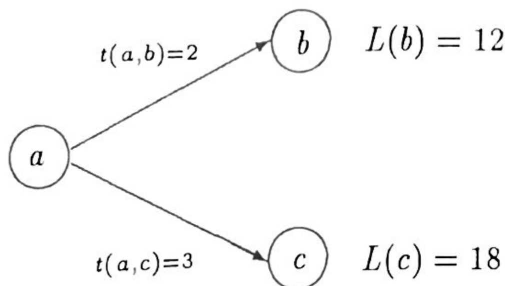
По този начин за $E(c)$ получаваме

$$E(c) = \max\{4,\ 5\} = 5.$$

От пример 6.3 е ясно, че в общия случай, в даден граф $G = (V, E)$, за събитието y

$$E(y) = \max_{x:(x,y)\in E}\{E(x) + t(x,y)\}. \tag{6.1}$$

ПРИМЕР 6.4. Да се определи най-късният срок за настъпване на събитието a в следния граф:



Събитието a предшества събитието b, следователно времето за настъпване на a не трябва да надхвърля

$$L(b) - t(a, b) = 10.$$

Едновременно с това, събитието a предшества и събитието c, поради което времето за настъпване на a не трябва да надхвърля и

$$L(c) - t(a, c) = 18 - 3 = 15.$$

Следователно, за $L(a)$ получаваме

$$L(a) = \min\{10,\ 15\} = 10.$$

От пример 6.4 е ясно, че в общия случай при даден граф $G = (V, E)$, за събитието x имаме

(6.2) $$L(x) = \min_{y:(x,y)\in E} \{L(y) - t(x, y)\}.$$

Разгледаните два примера, в частност (6.1) и (6.2), дават възможност да формулираме следните два алгоритъма.

3. Разчет на най-ранните срокове за настъпване на събития

ОПИСАНИЕ НА АЛГОРИТЪМА. *СТЪПКА 1.* Извършете топологическа сортировка на върховете в графа $G = (V, E)$, т.е. номерирайте събитията с числата 1, 2, ..., n ($n = |V|$), така, че за всяка операция (дъга) (x, y), да бъде изпълнено $x < y$.

Присвоете на върха с номер 1 най-ранен срок 0, т.е. $E(1) = 0$.

СТЪПКА 2. За всяко $y = 2,\ 3, ...,\ n$, определете

$$E(y) = \max_{x:(x,y)\in E} \{E(x) + t(x, y)\}.$$

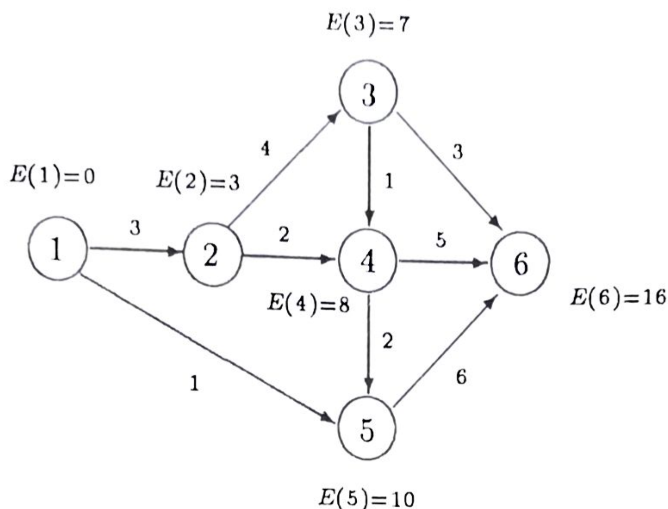
4. Разчет на най-късните срокове

ОПИСАНИЕ НА АЛГОРИТЪМА. *СТЪПКА 1.* Извършете топологична сортировка на върховете в графа $G = (V, E)$, т.е. номерирайте събитията с числата 1, 2,... n ($n = |V|$), така че за всяка операция (дъга) (x, y), да бъде изпълнено $x < y$. Положете $L(n) = T$, където T е времето за завършване на проекта.

СТЪПКА 2. За всяко $x = n - 1,\ n - 2, ...,\ 3,\ 2,\ 1$, определете

$$L(x) = \min_{y:(x,y)\in E} \{L(y) - t(x, y)\}.$$

ПРИМЕР 6.5. Да се изчислят най-ранните срокове за настъпване на събитията от следния мрежови график:



Черт. 2.4

След прилагане на алгоритъма за топологична сортировка, ще получим номерация на върховете, както е показано на графа.

СТЪПКА 1. $E(1) = 0$.

СТЪПКА 2. $E(2) = E(1) + t(1,2) = 0 + 3 = 3$.

$E(3) = E(2) + t(2,3) = 3 + 4 = 7$.

$$E(4) = \max\{E(3) + t(3,4),\ E(2) + t(2,4)\} = \max\{8,\ 5\} = 8.$$

$$E(5) = \max\{E(4) + t(4,5),\ E(1) + t(1,5)\} = \max\{10,\ 1\} = 10.$$

$$E(6) = \max\{E(3) + t(3,6),\ E(4) + t(4,6),\ E(5) + t(5,6)\} =$$

$$= \max\{10,\ 13,\ 16\} = 16.$$

Очевидно е сходството на алгоритъма за разчет на най-ранните срокове със задачата за търсене на най-дълъг път в ориентиран ацикличен граф. Маркиращите числа $E(x)$, получени в разчета, дават дължините на най-дългите пътища от началния връх 1 до върха x. В частност $E(n)$ дава дължината на най-дългия път от източника s до стока t. Ако замените операцията max в разчета за най-ранните срокове с операцията min, ще получите ли алгоритъм за търсене на най-кратък път между източника s и стока t в ориентиран ацикличен граф? (Има ли значение броят на източниците и стоковете?)

В графа от пример 6.5, $E(6) = 16$ е дължината на най-дългия път между върховете 1 и 6. Самият път, т.е. дъгите, които го образуват, могат да бъдат намерени с последователно връщане от последния връх към първия. Най-общо, започвайки от върха $x_j = n$, полагаме на всяка стъпка x_j равен на такъв връх v_i, за

който

$$E(v_i) = E(x_j) - t(v_i, x_j),$$

като правим това, докато не стигнем началния връх, т.е. $v_i = 1$.

В пример 6.5, това е пътят P

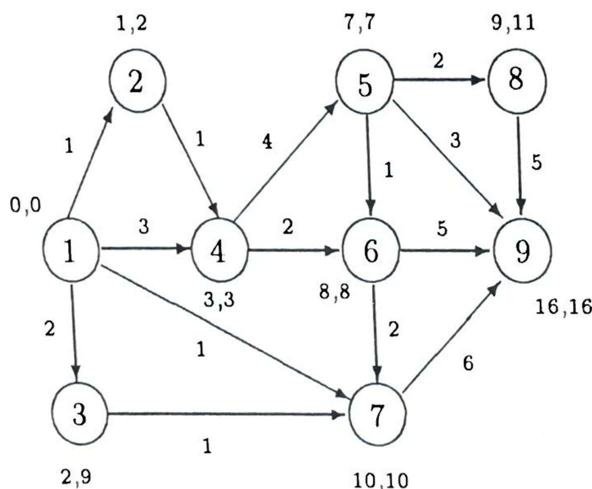
$$P: \ (1,2),\ (2,3),\ (3,4),\ (4,5),\ (5,6).$$

Ако теглата на дъгите интерпретираме като дължини, дължината на този път ще бъде равна на $E(6) = 16$.

ЗАДАЧА 6.1. За дадения по-долу мрежови график, да се намери:

а) разчет за най-ранните срокове $E(X)$;

б) разчет за най-късните срокове $L(X)$, при $L(9) = 16$.



Черт. 2.5

Решение: а) Една от възможните топологични номерации на събитията е тази от черт. 2.5.

СТЪПКА 1. $E(1) = 0$.

СТЪПКА 2. $E(2) = 0 + 1 = 1$.

$E(3) = 0 + 2 = 2$.

$$E(4) = \max\{1 + 1,\ 0 + 3\} = 3.$$

$$E(5) = 3 + 4 = 7.$$

$$E(6) = \max\{7 + 1,\ 3 + 2\} = 8.$$

$$E(7) = \max\{8 + 2,\ 0 + 1,\ 2 + 1\} = 10.$$

$$E(8) = 7 + 2 = 9.$$

$$E(9) = \max\{9 + 5,\ 7 + 3,\ 8 + 5,\ 10 + 6\} = 16.$$

Решение б): Условието $L(9) = E(9) = 16$ означава, че проектът трябва да бъде завършен възможно най-рано.

СТЪПКА 1. $L(9) = 16$.

СТЪПКА 2. $L(8) = L(9) - t(8,9) = 16 - 5 = 11$.

$L(7) = L(9) - t(7,9) = 16 - 6 = 10$.

$L(6) = \min\{L(9) - t(6,9),\ L(7) - t(6,7)\} = \min\{16 - 5,\ 10 - 2\} = 8$.

$L(5) = \min\{L(8) - t(5,8),\ L(9) - t(5,9),\ L(6) - t(5,6)\} =$

$= \min\{11 - 2,\ 16 - 3,\ 8 - 1\} = 7$.

$L(4) = \min\{L(6) - t(4,6),\ L(5) - t(4,5)\} = \min\{8 - 2,\ 7 - 4\} = 3$.

$L(3) = L(7) - t(3,7) = 10 - 1 = 9$.

$L(2) = L(4) - t(2,4) = 3 - 1 = 2$.

$L(1) = \min\{L(2) - t(1,2),\ L(4) - t(1,4),\ L(7) - t(1,7),\ L(3) - t(1,3)\} =$

$= \min\{2 - 1,\ 3 - 3,\ 10 - 1,\ 9 - 2\} = 0$.

Следователно, завършването на проекта към момент от време 16 води до необходимост да се стартира в момент от време нула.

ЗАДАЧА 6.2. За мрежовия график от пример 6.5 да се определи разчет на най-късните срокове $L(x)$, $x = 1,\ 2,\ 3,\ 4,\ 5$, при $L(6) = 16$.

Отг.: $L(x) = E(x)$ за $x = 1,\ 2,\ 3,\ 4,\ 5$ (Защо?)

АНАЛИЗ И КОМЕНТАРИ. (1). Най-ранният срок $E(x)$ за настъпване на събитието x може да се интерпретира като дължината на най-дългия път от началното събитие 1 до събитието x. Очевидно, разликата $L(n) - L(x)$ може да се интерпретира като дължина на най-дългия път от събитието x до крайното събитие n.

(2). Ако $L(n) \geq E(n)$, то за всяко събитие x

$$L(x) \geq E(x).$$

(3). Увеличението на крайния срок $L(n)$ за завършване на проекта с t единици води до аналогични увеличения на всички останали най-късни срокове $L(x)$, $x = 1,\ 2, \ldots,\ n - 1$.

(4). Алгоритмите за разчет на най-ранни и най-късни срокове са с една и съща сложност, тъй като и двата алгоритъма изискват една операция събиране (или изваждане) за всяка дъга и една операция търсене на максимум (или минимум).

(5). *Пълен резерв от време.* Всяка операция (етап) (x, y) започва да се изпълнява не по-рано от момента $E(x)$ и завършва не по-късно от момента $L(y)$. Следователно максималното време, което може да се отдели за нейното изпълнение, без това да влияе на своевременното завършване на целия проект, е $L(y) - E(x)$. Тогава максималното закъснение при изпълнението на тази операция, което е допустимо (без да води до закъснение на изпълнението на целия проект) е

$$L(y) - E(x) - t(x, y). \tag{6.3}$$

Например, в мрежовия график от черт. 2.5 операцията (5.9) започва да се изпълнява не по-рано от седмия ден и може да завърши не по-късно от шестнадесетия ден (моментите от време могат да се интерпретират като дни, седмици, часове и т.н.). Следователно максималното време, което може да се отдели за нейното изпълнение (без промяна на крайния срок за изпълнение на проекта), е $16 - 7 = 9$ дни. Тъй като времето за изпълнение на самата операция е 3 дни, допустимото закъснение при изпълнение на тази операция е $9 - 3 = 6$ дни.

Величината $L(y) - E(x) - t(x, y)$ се нарича *пълен резерв от време* за изпълнение на операцията (x, y).

(6). *Свободен резерв от време.* Нека сега поискаме операцията (x, y) да бъде изпълнена до момента $E(y)$. Тъй като нейното изпълнение може да започне не по-рано от момента $E(x)$, то $E(y) - E(x)$ е максималното време за нейното изпълнение, което не води до допълнителни времеви ограничения на последващите операции.

Тъй като времето за изпълнение на операцията (x, y) е $t(x, y)$, то величината

$$E(y) - E(x) - t(x, y) \tag{6.4}$$

определя максимално възможното закъснение, при изпълнение на тази операция (x, y), невлияещо върху изпълнението на последващите операции.

Величината (6.4) се нарича *свободен резерв от време* за изпълнение на операцията (x, y).

(7). Величините, определени в (6.3) и (6.4), са неотрицателни.

От (6.3) и (6.1) следва

$$L(y) - E(x) - t(x, y) = L(y) - [E(x) + t(x, y)] \geq L(y) - E(y).$$

Тъй като $L(y) \geq E(y)$ следва $L(y) - E(y) \geq 0$, т.е.

$$L(y) - E(x) - t(x, y) \geq 0.$$

От (6.4) и (6.1) следва, че

$$E(y) - E(x) - t(x, y) \geq 0.$$

(8). *Независим резерв от време.* Да допуснем сега, че операцията (x, y) започва да се изпълнява възможно най-късно, т.е. в момента $L(x)$ и завършва възможно най-рано, т.е. в момента $E(y)$. В този случай за изпълнение на операцията (x, y) може да се отдели не повече (т.е. максимум) от $E(y) - L(x)$ единици време, без това да води до допълнителни времеви ограничения на която и да е операция (етап) от проекта. Следователно величината

$$E(y) - L(x) - t(x, y) \tag{6.5}$$

дава максималното закъснение, допустимо при изпълнение на операцията (x, y), неводещо до времеви ограничения на която и да е друга операция. Тази величина (6.5) се нарича *независим резерв от време* за изпълнение на операцията (x, y).

Например, в мрежовия график от черт. 2.5 независимият резерв от време за изпълнение на операцията (3,7) е

$$E(7) - L(3) - t(3, 7) = 10 - 9 - 1 = 0.$$

Когато независимият резерв от време е отрицателен, това означава, че всяко закъснение при изпълнението на тази операция води до допълнителни времеви ограничения за изпълнението на други операции.

(9). От начина, по който са дефинирани величините (6.3), (6.4) и (6.5), и поради $L(x) \geq E(x)$, следва

$$\underset{\text{(пълен резерв)}}{L(y) - E(x) - t(x, y)} \geq \underset{\text{(свободен резерв)}}{E(y) - E(x) - t(x, y)} \geq \underset{\text{(независим резерв)}}{E(y) - L(x) - t(x, y)}. \tag{6.6}$$

За мрежовия график от черт. 2.5, стойностите на трите резерва съответно са:

	$L(y)-E(x)-t(x,y)$	$E(y)-E(x)-t(x,y)$	$E(y)-L(x)-t(x,y)$	
Операция	Пълен резерв	Свободен резерв	Независим резерв	Критичен път
(1,2)	$2-0-1=1$	$1-0-1=0$	$1-0-1=0$	
(1,4)	$3-0-3=0$	$3-0-3=0$	$3-0-3=0$	*
(1,7)	$10-0-1=9$	$10-0-1=9$	$10-0-1=9$	
(1,3)	$9-0-2=7$	$2-0-2=0$	$2-0-2=0$	
(2,4)	$3-1-1=1$	$3-1-1=1$	$3-2-1=0$	
(3,7)	$10-2-1=7$	$10-2-1=7$	$10-9-1=0$	
(4,5)	$7-3-4=0$	$7-3-4=0$	$7-3-4=0$	*
(4,6)	$8-3-2=3$	$8-3-2=3$	$8-3-2=3$	
(5,6)	$8-7-1=0$	$8-7-1=0$	$8-7-1=0$	*
(5,8)	$11-7-2=2$	$9-7-2=0$	$9-7-2=0$	
(5,9)	$16-7-3=6$	$16-7-3=6$	$16-7-3=6$	
(6,7)	$10-8-2=0$	$10-8-2=0$	$10-8-2=0$	*
(6,9)	$16-8-5=3$	$16-8-5=3$	$16-8-5=3$	
(7,9)	$16-10-6=0$	$16-10-6=0$	$16-10-6=0$	*
(8,9)	$16-9-5=2$	$16-9-5=2$	$16-11-5=0$	

Операция, на която пълният резерв от време е нула (от (6.6) следва, че и свободният и независимият резерв също са такива), се нарича *критична операция*. Това е операция, при изпълнението на която всяко закъснение води до закъснение на целия проект. Тези операции, които не са критични, допускат закъснения, непревишаващи пълния резерв, без това да влияе върху крайния срок за изпълнение на проекта. Ако $L(n) = E(n)$, всяка операция от най-дългия път между началното и крайното събитие на мрежовия график се явява *критична*, а пътят, състоящ се само от критични операции, се нарича *критичен път* ($E(n)$ е дължината на най-дългия път). Досега предполагахме, че времето за изпълнение на една операция е известно и е $t(x, y)$. В действителност трябва да се отчита, че продължителността за изпълнение на една операция е неопределена. Има разработен метод PERT (Project Evaluation Research Task, на други места в литературата Program Evaluation Review Technique), който повтаря изложения тук метод на критичния път CPM (Critical Path Method) с тази разлика, че времената са очаквани, а не детерми-

пирани. За изчисление на тези времена се използват три оценки [1]:

O — оптимистична оценка за времето на изпълнение;

P — песимистична оценка за времето на изпълнение;

R — реалистична оценка за времето на изпълнение.

Очакваното време за изпълнение на операцията се оценява като

$$\frac{O}{6} + \frac{P}{6} + \frac{4}{6}R.$$

Дисперсията на времето за изпълнение на операцията се задава с $\left(\frac{P-O}{6}\right)^2$. Прилагането на метода на критичния път в този случай води до резултати, които са очаквани — $E(x)$ е очакваният най-ранен срок, $L(x)$ е очакваният най-късен срок за изпълнение на събитието x, а $E(n)$ е очакваният най-ранен срок за изпълнение на проекта. Действителният най-ранен срок за изпълнение на проекта се счита за нормално разпределена случайна величина с математическо очакване $E(n)$ и дисперсия, равна на сумата от дисперсиите на операциите от най-дългия път между началното събитие и крайното (ако има няколко такива пътища, се взема пътят с максимална сума на дисперсиите). Това предположение дава възможност да се направят вероятностни оценки на реалното време за завършване на проекта. В метода PERT хипотезата за нормално разпределение на действителното време за завършване на проекта е толкова по-съдържателна, колкото времената за изпълнение на отделните операции са по-малко статистически зависими.

Друга особеност на метода PERT е, че той решава задачи за оптимиране, при които не се отчитат наличните ресурси, както и необходимостта от ресурси за всеки отделен етап. С помощта на метода се намира критичен път, след което се прави такова разпределение на ресурсите по етапи, което позволява закъснения само в етапи, непринадлежащи на критичния път. Задачата, при която се отчитат ограниченията на ресурсите още на стадий планиране, е доста по-сложна и трудна за решаване.

5. Проекти с минимална стойност

От досегашните разглеждания става ясно, че при управление изпълнението на даден проект, проблемът е в оптималното разпределение на закъсненията върху некритичните операции (закъсненията при критичните операции са недопустими — нарушават своевременното изпълнение на проекта). Да предпо-

ложим, че стойността за изпълнение на всяка операция (x, y) е

$$(6.7) \qquad K(x,y) - k(x,y).t(x,y),$$

където $K(x,y)$ е произволна константа, а $k(x,y)$ е положителен коефициент, т.е. с увеличение на времето за изпълнение на операцията с единица, стойността ѝ се намалява с $k(x,y)$. Освен това, нека за времето на изпълнение на дадена операция имаме ограниченията

$$(6.8) \qquad r(x,y) \le t(x,y) \le s(x,y).$$

Предположението за линейност на функцията (6.7) в много случаи е резонно.

И така проблемът е какво време $t(x,y)$ да определим за изпълнение на операцията (x,y), така че стойността за изпълнение на проекта да е минимална, като се спази даденият краен срок T за изпълнение на проекта.

Това означава да се намери оптимално време $p(x)$ за възникване на всяко събитие x, така че да са изпълнени условията:

$$(6.9) \qquad p(1) = 0,\ p(n) = T \text{ и } p(y) - p(x) \ge r(x,y).$$

Времето $t(x,y)$ за изпълнение на всяка операция ще се избира възможно максимално, т.е.

$$(6.10) \qquad t(x,y) = \min\{s(x,y),\ p(y) - p(x)\}.$$

Като задача на линейното оптимиране, задачата за определяне продължителността на всяка операция при минимална стойност на проекта има следния модел:

$$(6.11) \qquad \sum_{(x,y)} [K(x,y) - k(x,y).t(x,y)] \to \min$$

при условия

$$(6.12) \qquad p(n) - p(1) \le T,$$

$$(6.13) \qquad p(y) - p(x) - t(x,y) \ge 0, \text{ за } \forall (x,y),$$

$$(6.14) \qquad r(x,y) \le t(x,y), \text{ за } \forall (x,y),$$

$$t(x,y) = s(x,y), \text{ за } \forall(x,y). \tag{6.15}$$

Ще дадем начин за решаване на тази задача, предложен от Фалкерсон ([10]), който използва разгледания вече алгоритъм на дефекта за намиране на поток с минимална стойност.

За целта в мрежовия график $G = (V, E)$ се правят следните промени, за да се получи модифициран мрежови график $G_1 = (V_1, E_1)$.

1. Въвежда се възвратна дъга $(n, 1) \in E_1$, с цена $a(n, 1) = T$ и пропускателна способност $c(n, 1) = \infty$.

2. Всяка дъга (операция) (x, y) се заменя с две дъги

$$(x,y)_1 \in E_1 \quad \text{и} \quad (x,y)_2 \in E_1$$

съответно с цени

$$a(x,y)_1 = -s(x,y) \quad \text{и} \quad a(x,y)_2 = -r(x,y),$$

и пропускателни способности

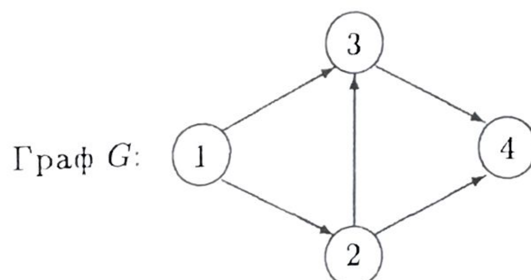
$$c(x,y)_1 = k(x,y) \quad \text{и} \quad c(x,y)_2 = \infty.$$

▷ *ТЕОРЕМА 6.2. Ако $P(x)$ са маркиращите числа на върховете, получени след прилагане на алгоритъма на дефекта за графа G_1, то*

$$p(x) = P(1) - P(x)$$

са оптималните времена (моменти) за настъпване на събитието x. ◁

ПРИМЕР 6.6. [1] Да се определят оптималните моменти за настъпване на събитията и оптималните продължителности за изпълнение на операциите в следния мрежов график:

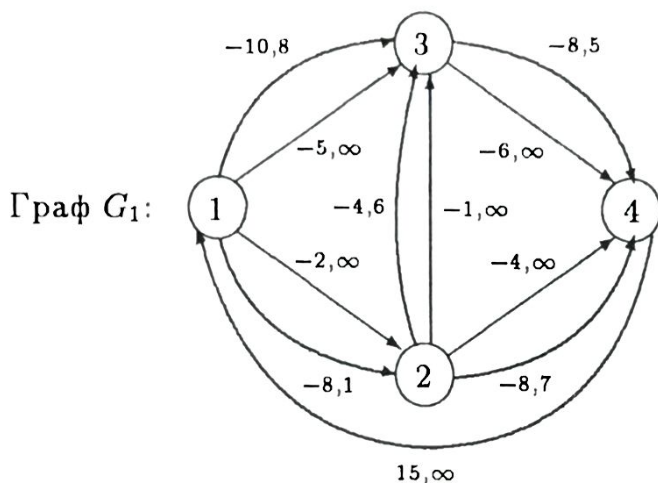


$$
\begin{aligned}
2 \le t(1,2) &\le 8, & k(1,2) &= 1,\\
5 \le t(1,3) &\le 10, & k(1,3) &= 8,\\
1 \le t(2,3) &\le 4, & k(2,3) &= 6,\\
4 \le t(2,4) &\le 8, & k(2,4) &= 7,\\
6 \le t(3,4) &\le 8, & k(3,4) &= 5.
\end{aligned}
$$

Срок за изпълнение на проекта, $T = 15$.

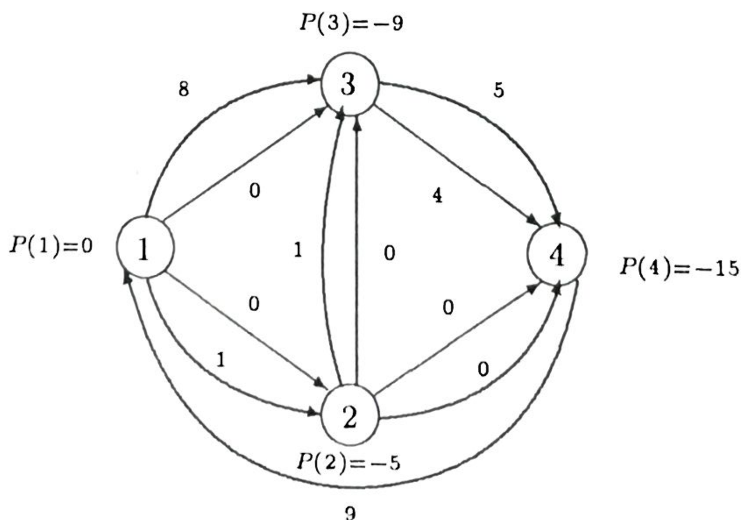
Модифицираме дадения мрежови график, както бе указано по-горе. Получаваме следния мрежови график G_1.

Граф G_1:



В този мрежови график първите числа са цени, а вторите са пропускателни способности.

За модифицирания граф G_1 се прилага алгоритъмът на дефекта за намиране на поток с минимална стойност. Резултатът от прилагането на този алгоритъм е следният:



В последния получен граф са показани потоците в дъгите и двойнствените променливи $P(x)$ (маркиращите числа) на всеки от върховете, след като алгоритъмът на дефекта е приключил своята работа. Това дава възможност с помощта на теорема 6.2 и (6.10) да определим оптималните моменти за настъпване на всяко събитие и оптималната продължителност за изпълнение на операциите.

Оптимални моменти за настъпване на събитието:

$$p(1) = P(1) - P(1) = 0,$$
$$p(2) = P(1) - P(2) = 0 - (-5) = 5,$$
$$p(3) = P(1) - P(3) = 0 - (-9) = 9,$$
$$p(4) = P(1) - P(4) = 0 - (-15) = 15.$$

Оптимална продължителност за изпълнение на операциите:

$$t(1,2) = \min\{s(1,2),\ p(2) - p(1)\} = \min\{8,\ 5 - 0\} = 5,$$
$$t(1,3) = \min\{s(1,3),\ p(3) - p(1)\} = \min\{10,\ 9 - 0\} = 9,$$
$$t(2,3) = \min\{s(2,3),\ p(3) - p(2)\} = \min\{4,\ 9 - 5\} = 4,$$
$$t(2,4) = \min\{s(2,4),\ p(4) - p(2)\} = \min\{8,\ 15 - 5\} = 8,$$
$$t(3,4) = \min\{s(3,4),\ p(4) - p(3)\} = \min\{8,\ 15 - 9\} = 6.$$

6. Обобщени мрежови графици

Сега накратко и информативно ще направим някои бележки, свързани с възможността разглежданите досега мрежови графици да бъдат обобщени.

Твърде често практиката поставя проблеми, които не могат адекватно да бъдат описани с графиците, които разглеждахме досега. Например, невинаги е задължително да се изпълняват всички операции на даден проект. Представете си, че в една автомобилна компания има мрежови график, описващ всички операции по сглобяването на автомобила и отношението предшестване.

Ясно е, че операции от тип "екстри" (по желание на клиента) като монтаж на въздушна възглавница, допълнителен резервоар, допълнителна арматура на купето, монтаж на бордови компютър и т.н. не са задължителни.

Освен това, досегашното предположение в мрежовия график всяка операция, започваща след събитието x, да се изпълнява при условие, че са изпълнени всички предшестващи това събитие операции, не винаги е съдържателно ограничение. Много често изпълнението на поне една предшестваща събитието x операция е достатъчно основание за извършването на операции, последващи събитието x. Това е причината, която налага дефинирането на *обобщени мрежови графици*. Това са мрежови графици, в които съществува диференциация на върховете, т.е. разглеждат се върхове от различен тип, които се наричат *разрешаващи възли* (РВ). Всеки РВ се характеризира с условията, които се налагат на дъгите (операциите), инцидентни с него.

а) *конюнктивен вход* — събитието, съответстващо на дадения РВ, се счита за настъпило, ако са изпълнени *всички влизащи в РВ операции*;

б) *дизюнктивен вход* — събитието, съответно на дадения РВ, се счита за настъпило, ако *поне една от влизащите в РВ операции е изпълнена*;

в) *алтернативен вход* — събитието, съответстващо на дадения РВ, се счита за настъпило, ако е изпълнена *точно една* от влизащите в РВ операции.

На операциите, излизащи от РВ, е възможно да бъдат наложени следните две ограничения:

а) *детерминиран изход* — след сбъдване на събитието, съответстващо на даден РВ, се изпълняват *всички излизащи от възела операции*;

б) *вероятностен изход* — след сбъдване на събитието, съответстващо на дадения РВ, се изпълнява *точно една излизаща от възела операция*.

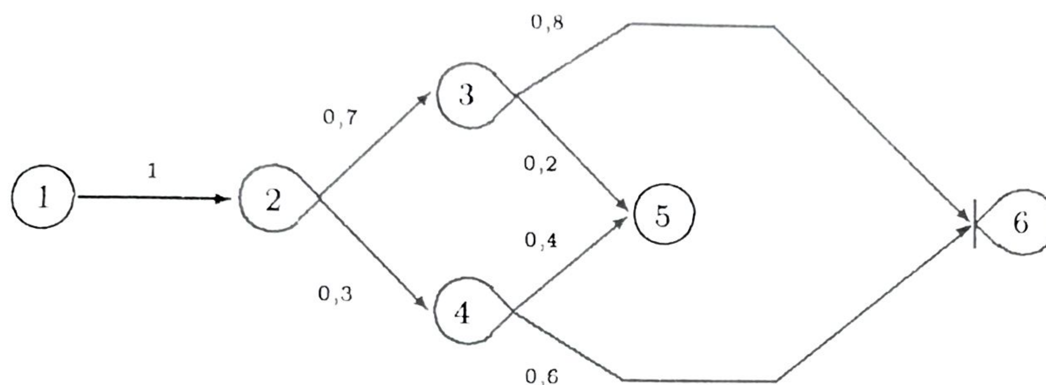
Очевидно, при така наложените ограничения на инцидентните с възела операции съществуват шест различни типа разрешаващи възли (РВ). Графично те могат да се изобразят така:

Вход / Изход	конюнктивен	дизюнктивен	алтернативен
детерминиран			
вероятностен			

За разлика от досегашните мрежови графици, в които се задаваха само времената $t(x, y)$ за изпълнение на всяка операция (x, y), при обобщените графици се задава и вероятността $p(x, y)$

за изпълнение на всяка операция (x, y) — това е вероятността след появяване на събитието, съответстващо на разрешаващия възел x, да бъде изпълнена операцията (x, y). При възли x с детерминиран изход, вероятността $p(x, y) = 1$ и операцията (x, y) задължително се изпълнява. При възли x с вероятностен изход, сумата от вероятностите на излизащите от x операции не трябва да надминава 1.

ПРИМЕР 6.7. Да разгледаме следния обобщен мрежови график



Разрешаващият възел 1 има детерминиран изход, следователно за операцията (1,2), $p(1, 2) = 1$, т.е. операцията трябва да бъде изпълнена.

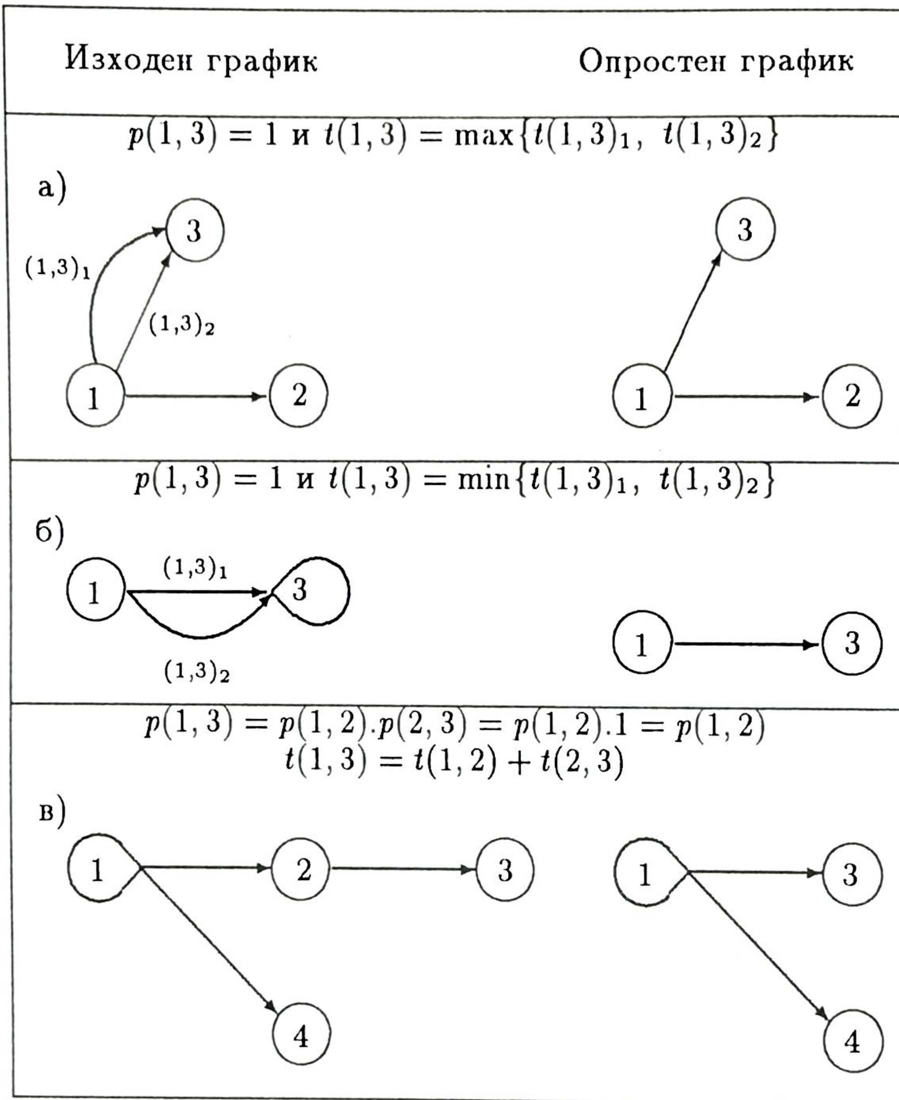
Възлите 2, 3, 4 и 5 имат конюнктивен вход, т.е. съответните на тях събития се сбъдват, ако са изпълнени всички входящи в тях операции. След настъпване на всяко от събитията 2, 3 и 4, които са с вероятностен изход, се изпълнява точно една излизаща от събитието операция. Например, след настъпване на 3 се изпълнява с вероятност $0,8$ операцията $(3,6)$ или с вероятност $0,2$ операцията $(3,5)$, но само една от тези две операции.

Събитието, съответстващо на възела 6, настъпва точно когато едната от операциите (3,6) и (4,6) е изпълнена (то не настъпва, когато едновременно двете операции са изпълнени или и двете не са изпълнени).

В досегашните мрежови графици събитията рано или късно настъпваха. При обобщените графици това не е задължително, поради възможността някои операции да не се изпълняват. Това означава, че е възможно проектът да не завършва в разрешаващ възел, а да завършва след изпълнение на някоя операция. В обобщения мрежови график от пример 6.7 се вижда, че ако бъдат изпълнени операциите (3,6) и (4,6), събитието, съответстващо на възела 6, няма да настъпи.

Възможни са опростявания в обобщения мрежови график, които ще илюстрираме със следния пример:

ПРИМЕР 6.8.



Очевидно, опростените графици са еквивалентни на изходните и съдържат по-малко операции.

Съществуването на различни възможности за завършване на проекта прави интересен въпроса за познаване на вероятностите за сбъдването на събитията в обобщения мрежови график или вероятностите за реално изпълнение на операциите, както и математическото очакване на времето за появяване на настъпващи събития. Изчисляването на тези вероятности и математическото очакване е свързано с редица трудности, при наличието на възли с вероятности изходи. Съществуването на такъв възел води до статистическа зависимост, което прави неприложими традиционните правила за изчисление, предполагащи статистическа независимост. Например, за сбъдване на събитието 5 от пример 6.7 е необходимо да бъдат изпълнени и двете операции (3,5) и (4,5). Вероятността за сбъдването на операцията (3,5)

е $0,7.0,2 = 0,14$, а вероятността да бъде изпълнена операцията (4,5) е $0,3.0,4 = 0,12$. На пръв поглед, вероятността за сбъдване на събитието 5 "е" $0,14.0,12 = 0,0168$. В действителност обаче това не е така. Събитието 5 никога не настъпва, защото операциите (3,5) и (4,5) не се явяват статистически независими. Тези две операции (3,5) и (4,5) се изпълняват само когато едновременно са изпълнени операциите (2,3) и (2,4), което е невъзможно поради това, че събитието 2 е с вероятностен изход, т.е. след неговото настъпване се изпълнява само една от операциите (2,3) и (2,4). Дори опитите за промяна в дефинирането на разрешаващ възел с вероятностен изход като възел, чието сбъдване води до изпълнението на няколко излизащи от него операции, водят до изчислителни трудности. Тези трудности отново са свързани с отсъствие на статистическа независимост.

Накрая, при обобщените мрежови графици ситуацията се усложнява допълнително и от възможността да съществуват цикли — в някои проекти има операции, изпълнението на които трябва да се повтаря докато не бъдат изпълнени правилно (операцията полагане на покрития в строителството и автомобилостроенето може да се наложи да бъде извършена многократно).

Повече информация за обобщени мрежови графици може да намерите в [32], [33], [34] и др.

2.7. Разполагане на обекти

В много случаи практиката поставя следния тип проблеми: как да се разположат обекти (центрове за медицинска и пътна помощ, заводи, складове, телефонни централи, магазини и др.) в структурата на даден град, област или държава, така че това разположение да бъде оптимално. Критерият за оптималност може да се състои в минимизиране на разстоянието или времето за достигане до съответния обслужващ обект (или обекти).

Най-общо казано, се налага да се решават задачи за "най-добро" разположение в графи и мрежи. Ние ще разгледаме задачи, при които обектът се разполага във върховете или върху дъгите (ребрата) на графа.

Една възможна такава задача е този обект да се разположи така, че да се минимизира разстоянието до най-отдалечения връх на графа – оптимизация на "най-лошия вариант".

Модификация на тази задача е задачата, при която във върховете или върху дъгите на графа трябва да се разположат няколко обекта. При тази задача най-отдалеченият връх на графа трябва да се намира поне от един пункт за обслужване

на минимално възможното разстояние – оптимизация на разстоянието от произволен връх на графа до най-близкия пункт за обслужване.

Задачи от този тип се наричат *минимаксни задачи за разположение.* Местата за разположение на пунктовете за обслужване в тези задачи се наричат *центрове* на графа.

В други задачи за разполагане на обекти се налага да се минимизира сумата на разстоянията от върховете на графа до обекта за обслужване. Типична в това отношение е задачата за разполагане на складове в граф, на който върховете представляват потребителите, а дъгите (ребрата) са съответната пътна мрежа. Задачите от този тип се наричат *минисумарни задачи за разположение.* При това целевата функция при тези задачи може да е не просто сума от разстоянията, а сума от различни функции от разстоянията. Местата на разположение на пунктовете за обслужване при тези задачи се наричат *медиани* на графа.

Решаването на задачи за разположение на центрове и медиани в граф очевидно налага да се въведат формални и строги определения, свързани с описание на точките от дъгите на графа и възможните различни разстояния в него. Ще направим това.

	Нека (i,j) е произволна дъга с дължина $a(i,j) > 0$ и f е число, $0 \leq f \leq 1$. Точката от дъгата (i,j), отстояща на разстояние $fa(i,j)$ от върха i и $(1-f).a(i,j)$ от върха j, се нарича $f-$точка.

От даденото определение е ясно, че можем да говорим за *нулева точка* на дъгата (i,j) – такава се явява върхът i и единична точка на дъгата (i,j) – такава се явява върхът j. С други думи, върховете на графа можем да разглеждаме като точки от дъгите, а точките, които не са върхове, се наричат вътрешни точки.

Ще дефинираме няколко типа *разстояния в граф.*

РАЗСТОЯНИЕ ОТ ТИП "ВРЪХ-ВРЪХ" (ВВ). Да означим с d_{ij} дължината на най-краткия път от върха i до върха j. Както вече изяснихме, тези разстояния са елементите на матрицата D^n с размери $n \times n$, получена след прилагането на алгоритмите на Флойд и Данциг за търсене на най-кратки пътища между

всички върхове на графа.

РАЗСТОЯНИЕ ОТ ТИП "ТОЧКА-ВРЪХ" (ТВ). Да означим с $d\,(f$ от $(r,s),j)$ дължината на най-краткия път от f–точката на дъгата (реброто) (r,s) до върха j. Тази величина, ще наричаме *разстояние от тип точка-връх*.

а) Нека (r,s) е ребро. Тогава

$$(7.1)\quad d(f \text{ от } (r,s),j) = \min\{fa(r,s)+d_{rj},\ (1-f)a(r,s)+d_{sj}\},$$

т.е. за разстояние се избира по-малкото от следните две разстояния: разстоянието от f–точката до върха r плюс най-краткия път между върховете r и j или разстоянието от f–точката до върха s плюс разстоянието от s до j.

б) Ако (r,s) е дъга, т.е. обходът е допустим само по посока от r към s

$$(7.2)\qquad d(f \text{ от } (r,s),j) = (1-f)a(r,s)+d_{sj}.$$

От начина, по който дефинирахме разстоянието "точка-връх" е ясно, че за неговото намиране е достатъчно да са известни дължините на дъгите и елементите на матрицата D^n от алгоритъма на Флойд и Данциг.

РАЗСТОЯНИЕ ОТ ТИП "ВРЪХ-ДЪГА (РЕБРО)" (ВД). Да разгледаме минималните разстояния от върха j до всяка точка на дъгата (реброто) (r,s). За някоя точка от дъгата (реброто) това разстояние приема максимална стойност. Обозначава се с $d'(j,(r,s))$ и се нарича *разстояние от тип връх-дъга*.

а) Нека (r,s) е ребро. Тогава очевидно има два маршрута за движение от върха j до f-точката на реброто (r,s) — единият през върха r, а другият през върха s. Избира се по-краткият от тези два маршрута. Очевидно е вярно, че ако двата маршрута са с различна дължина, ще съществуват съседни точки на f-точката от реброто (r,s), които са по-отдалечени от върха j. Следователно за най-отдалечената от върха j, f-точка на реброто (r,s), двете разстояния до върха j ще бъдат равни. За сумата на тези разстояния имаме

$$d_{jr}+fa(r,s)+d_{js}+(1-f)a(r,s) = d_{jr}+d_{js}+a(r,s),$$

откъдето следва

$$(7.3) \qquad d'(j,(r,s)) = \frac{d_{jr} + d_{js} + a(r,s)}{2}$$

б) Нека (r,s) е дъга, т.е. произволна f-точка на дъгата (r,s) се достига само през върха r. Ясно е, че в този случай най-отдалечената от върха j точка на дъгата (r,s) се явява точката, най-близка до s, т.е. f-точката, за която $f = 1$. С други думи

$$(7.4) \qquad d'(j,(r,s)) = d_{jr} + a(r,s).$$

От казаното следва, че разстоянията от типа връх-дъга могат да се изчислят с помощта на (7.3) и (7.4), стига да са известни дължините на дъгите на графа и разстоянията от типа връх-връх, които са елементи на матрицата D^n от алгоритъма на Флойд и Данциг. Очевидно разстоянията от тип връх-дъга могат да се разположат в някаква матрица D_1 с размери $n \times m$, където n е броят на върховете, а m е броят на дъгите. Всеки елемент (j,t) на матрицата D_1 ще представлява разстоянието от j-тия връх до t-тата дъга.

Ще резюмираме дадените определения за прегледност в таблица.

Означение	Тип	Пресмятане
$a(i,j)$	Дължина на дъга	Известна
d_{ij}	Разстояние "връх-връх" (ВВ)	Алгоритъм на Флойд или Данциг
$d(f$ от $(r,s), j)$	Разстояние "точка-връх" (ТВ)	Формули (7.1) и (7.2)
$d'(j,(r,s))$	Разстояние "връх-дъга" (ВД)	Формули (7.3) и (7.4)

Могат да се дефинират и разстояния от типа "точка-дъга". Ние ще се ограничим с въведените по-горе понятия.

Да означим максималното разстояние от върха i до върховете на графа (разстоянието от i до най-отдалечения връх) с $MAXBB(i)$, т.е.

$$(7.5) \qquad MAXBB(i) = \max_j \{d_{ij}\}.$$

Център на графа G се нарича такъв връх x, за който

$$MAXBB(x) = \min_i \{MAXBB(i)\},$$

т.е. всеки връх на графа, разстоянието от който до най-отдалечения от него връх е минимално.

Да означим максималното измежду всички разстояния от f-точката на дъгата (r, s) до върховете на графа (разстоянието от f-точката до най-отдалечения връх) с $MAXTB(f \text{ от } (r,s))$, т.е.

$$MAXTB(f \text{ от } (r,s)) = \max_j \{d(f \text{ от } (r,s), j)\}. \tag{7.6}$$

Абсолютен център на графа G се нарича такава f-точка от произволна дъга (r, s), за която

$$MAXTB(f \text{ от } (r,s)) = \min_{f \text{ от } (u,v) \in P} \{MAXTB(f \text{ от } (u,v))\},$$

където P е множеството от всички точки на графа; т.е. всяка точка, разстоянието от която до най-отдалечения връх на графа е минимално.

Да означим максималното измежду всички разстояния от върха j до дъгите на графа (разстоянието от върха j до най-отдалечената от него точка на графа) с $MAXBД(j)$, т.е.

$$MAXBД(j) = \max_{(r,s)} \{d'(j, (r,s))\}. \tag{7.7}$$

Главен център на графа G се нарича такъв връх x, за който

$$MAXBД(x) = \min_j\{MAXBД(j)\},$$

т.е. всеки връх, разстоянието от който до най-отдалечената точка в графа е минимално.

Аналогично може да се въведе и понятието *главен абсолютен център* като точка, разстоянието от която до най-отдалечената точка е минимално. За целта обаче трябва да се дефинира разстояние между точка и дъга, което както вече споменахме, излиза извън рамките на нашите разглеждания.

По аналогия с трите типа центрове (разположения), които дефинирахме по-горе, можем да дадем определения и за медиана, главна медиана и абсолютна медиана в граф. Това са места (точки) в графа за разполагане на обекти, когато се решават минисумарни задачи за разположения.

Да означим сумарното разстояние от върха i до всички върхове на графа със

$$CBB(i) = \sum_j d_{ij}.$$

Медиана в графа G се нарича такъв връх x, за който

$$CBB(x) = \min_i\{CBB(i)\},$$

т.е. такъв връх x, за който сумата от разстоянията от него до останалите върхове на графа е минимална.

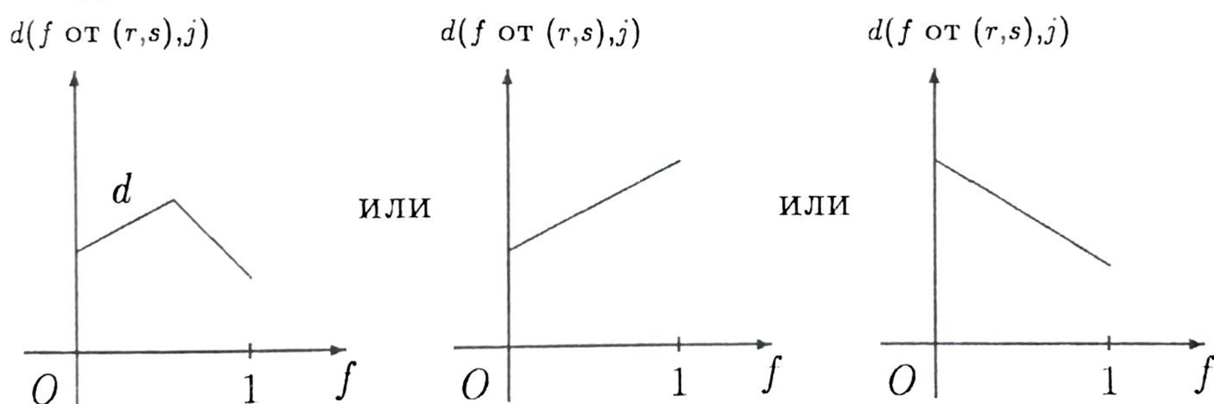
Абсолютна медиана в графа G се нарича такава точка от дъга на графа, за която сумарното разстояние от нея до всички върхове на графа е минимално.

> ▷ *ТЕОРЕМА 7.1. В графа винаги съществува връх, който се явява абсолютна медиана (всяка медиана в същото време се явява и абсолютна медиана).*

Доказателство: Да означим със $CTB(f \text{ от } (r,s))$ сумарното разстояние от f-точката на дъгата (r,s) до всички върхове на графа, т.е.

$$CTB(f \text{ от } (r,s)) = \sum_j d(f \text{ от } (r,s), j). \tag{7.8}$$

В (7.8) имаме сума на функции (на f) от вида $d(f \text{ от } (r,s), j)$, дефинирани със (7.1) и (7.2). Очевидно графиката на всяка такава функция е от вида



С други думи, функциите $d(f \text{ от } (r,s), j)$ са изпъкнали и минималната стойност очевидно се постига в крайните точки, т.е. или при $f = 0$ или при $f = 1$. Тъй като функцията (7.8) е сума от изпъкнали функции, тя също е изпъкнала и достига своя минимум или при $f = 0$ или при $f = 1$.

Оттук следва, че нито една вътрешна точка от дъгата (r,s) не може да бъде абсолютна медиана. Такава се явява един от крайните върхове на дъгата (реброто) (r,s). С това теоремата е доказана. ◁

От доказаната теорема следва, че не се налага да се търсят специални методи за намиране на абсолютна медиана — достатъчно е да се изследват само върховете на графа.

Да означим сумарното разстояние от върха i до дъгите на

графа със

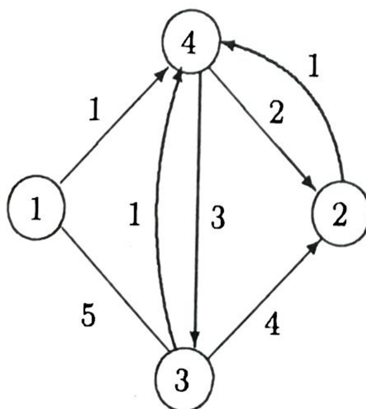
$$СВД(i) = \sum_{(r,s)} (d'(i,(r,s))).$$

Да припомним освен това, че под разстояние от връх до дъга се разбира максималното измежду разстоянията от върха до точките на дъгата.

> *Главна медиана на графа G* се нарича такъв връх x, за който
>
> $$СВД(x) = \min_i \{СВД(i)\},$$
>
> т.е. връх x, за който сумарното разстояние от него до всяка дъга е минимално.

1. Търсене на центрове

ПРИМЕР 7.1. Да се определи *центърът* на следния граф (т.е. да се намери онзи връх x на графа, за който разстоянието от него до най-отдалечения връх на графа е минимално):



С алгоритъма на Флойд се намира матрицата D^4, чийто елементи d_{ij} представляват дължините на най-кратките пътища между i-тия и j-тия връх.

$$D^4 = \begin{pmatrix} 0 & 3 & 4 & 1 \\ 9 & 0 & 4 & 1 \\ 5 & 3 & 0 & 1 \\ 8 & 2 & 3 & 0 \end{pmatrix}.$$

Тогава

$$MAXBB(1) = \max\{0,\ 3,\ 4,\ 1\} = 4$$

$$MAXBB(2) = \max\{9,\ 0,\ 4,\ 1\} = 9$$

$$MAXBB(3) = \max\{5,\ 3,\ 0,\ 1\} = 5$$

$$MAXBB(4) = \max\{8,\ 2,\ 3,\ 0\} = 8.$$

Следователно $\min_i\{MAXBB(i)\} = \min\{4,\ 9,\ 5,\ 8\} = 4$, т.е. върхът с номер 1 се явява център на графа, като разстоянието от него до най-отдалечения връх на графа е 4.

ПРИМЕР 7.2. Да се определи *главният център* на графа от пример 7.1 (т.е. върха x, разстоянието от който до най-отдалечената точка на дъгите на графа е минимално).

Построяваме матрицата D_1 с размери 4×7. Елементите на матрицата D_1 изчисляваме с помощта на формулите (7.3), (7.4) и елементите на матрицата D^4 от пример 7.1. Получаваме:

$$d'(1,(1,3)) = \tfrac{0+4+5}{2} = \tfrac{9}{2}$$
$$d'(1,(1,4)) = 0 + 1 = 1$$
$$d'(1,(3,4)) = 4 + 1 = 5$$
$$d'(1,(4,3)) = 1 + 3 = 4$$
$$d'(1,(4,2)) = 1 + 2 = 3$$
$$d'(1,(2,4)) = 3 + 1 = 4$$
$$d'(1,(3,2)) = 4 + 4 = 8$$

$$d'(2,(1,3)) = \tfrac{9+4+5}{2} = 9$$
$$d'(2,(1,4)) = 9 + 1 = 10$$
$$d'(2,(3,4)) = 4 + 1 = 5$$
$$d'(2,(4,3)) = 1 + 3 = 4$$
$$d'(2,(4,2)) = 1 + 2 = 3$$
$$d'(2,(2,4)) = 0 + 1 = 1$$
$$d'(2,(3,2)) = 4 + 4 = 8$$

$$d'(3,(1,3)) = \tfrac{5+0+5}{2} = 5$$
$$d'(3,(1,4)) = 5 + 1 = 6$$
$$d'(3,(3,4)) = 0 + 1 = 1$$
$$d'(3,(4,3)) = 1 + 3 = 4$$
$$d'(3,(4,2)) = 1 + 2 = 3$$
$$d'(3,(2,4)) = 3 + 1 = 4$$
$$d'(3,(3,2)) = 0 + 4 = 4$$

$$d'(4,(1,3)) = \tfrac{8+3+5}{2} = 8$$
$$d'(4,(1,4)) = 8 + 1 = 9$$
$$d'(4,(3,4)) = 3 + 1 = 4$$
$$d'(4,(4,3)) = 0 + 3 = 3$$
$$d'(4,(4,2)) = 0 + 2 = 2$$
$$d'(4,(2,4)) = 2 + 1 = 3$$
$$d'(4,(3,2)) = 3 + 4 = 7$$

Матрицата D_1 е следната

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{ccccccc} (1,3) & (1,4) & (3,4) & (4,3) & (4,2) & (2,4) & (3,2) \\ \hline 9/2 & 1 & 5 & 4 & 3 & 4 & 8 \\ 9 & 10 & 5 & 4 & 3 & 1 & 8 \\ 5 & 6 & 1 & 4 & 3 & 4 & 4 \\ 8 & 9 & 4 & 3 & 2 & 3 & 7 \end{array}.$$

$$MAXBД(1) = \max\{9/2,\ 1,\ 5,\ 4,\ 3,\ 4,\ 8\} = 8$$

$$MAXBД(2) = \max\{9,\ 10,\ 5,\ 4,\ 3,\ 1,\ 8\} = 10$$

$$MAXBД(3) = \max\{5,\ 6,\ 1,\ 4,\ 3,\ 4,\ 4\} = 6$$

$$MAXBД(4) = \max\{8,\ 9,\ 4,\ 3,\ 2,\ 3,\ 7\} = 9$$

Следователно

$$\min\{MAXBД(i)\} = \min\{8,\ 10,\ 6,\ 9\} = 6,$$

т.е. върхът 3 се явява главен център на графа.

Намирането на абсолютен център на графа (точка от дъга, разстоянието от която до най-отдалечения връх на графа е минимално) се извършва по метод, предложен в [35]. Ние няма да разглеждаме този въпрос.

2. Търсене на медиани

ПРИМЕР 7.3. Да се намери *медианата на графа* от пример 7.1.

Медианата е такъв връх x, на който сумарното разстояние от него до всички останали върхове на графа е минимално, т.е.

$$CBB(x) = \min_i\{CBB(i)\}.$$

Матрицата D^4, която се получава след прилагане на алгоритъма на Флойд, задава най-кратките разстояния от типа "връх-връх". Следователно сумата от елементите на i-тия ред на тази матрица дава сумарното разстояние от i-тия връх до останалите върхове на графа, т.е. $CBB(i)$. Оттук медианата е такъв връх, на който съответният ред от матрицата D^4 има минимална сумарна стойност.

Тъй като

$$D^4 = \begin{pmatrix} 0 & 3 & 4 & 1 \\ 9 & 0 & 4 & 1 \\ 5 & 3 & 0 & 1 \\ 8 & 2 & 3 & 0 \end{pmatrix},$$

имаме

$$CBB(1) = 0 + 3 + 4 + 1 = 8$$

$$CBB(2) = 9 + 0 + 4 + 1 = 14$$

$$CBB(3) = 5 + 3 + 0 + 1 = 9$$

$$CBB(4) = 8 + 2 + 3 + 0 = 13.$$

Следователно $\min\{CBB(i)\} = \min\{8,\ 14,\ 9,\ 13\} = 8$, т.е. върхът 1 се явява медиана на този граф, като сумарното разстояние от него до върховете на графа е 8.

ПРИМЕР 7.4. Да се намери *главната медиана на графа* от пример 7.1.

Да припомним, че главната медиана е такъв връх x, за който сумата от разстоянията от този връх до всяка от дъгите е минимална (разстояние от връх до дъга е максималното разстояние от върха до точките на дъгата), т.е.

$$CBД(x) = \min_i\{CBД(i)\}.$$

Елементите от i-тия ред, $1 \leq i \leq 4$ на матрицата D_1, получена в пример 7.2, даваха разстоянията от върха i до всяка от дъгите на графа. Следователно, сумата от елементите в i-тия ред на матрицата D_1 е равна на $CBД(i)$, т.е. главната медиана, това е онзи връх на графа, чийто съответен ред от матрицата D_1 има минимална сумарна стойност.

$$D_1 = \begin{pmatrix} 9/2 & 1 & 5 & 4 & 3 & 4 & 8 \\ 9 & 10 & 5 & 4 & 3 & 1 & 8 \\ 5 & 6 & 1 & 4 & 3 & 4 & 4 \\ 8 & 9 & 4 & 3 & 2 & 3 & 7 \end{pmatrix}.$$

$$CBД(1) = 9/2 + 1 + 5 + 4 + 3 + 4 + 8 = 34,5$$

$$CBД(2) = 9 + 10 + 5 + 4 + 3 + 1 + 8 = 40$$

$$CBД(3) = 5 + 6 + 1 + 4 + 3 + 4 + 4 = 27$$

$$CBД(4) = 8 + 9 + 4 + 3 + 2 + 3 + 7 = 36.$$

Следователно

$$\min_i\{CBД(i)\} = \min\{34,5;\ 40;\ 27;\ 36\} = 27,$$

т.е. върхът 3 се явява главна медиана на този граф, като минималното сумарно разстояние е 27.

2.8. Оптимални сдвоявания и покрития

Ще продължим изследванията, започнати в параграф 1.5 на предишната глава.

Множество от ребра в графа G такова, че всеки връх на графа е инцидентен с не повече от едно ребро от множеството, се нарича *сдвояване (съчетание)*.

Множество от ребра в графа G такова, че всеки връх на графа е инцидентен с поне едно ребро от това множество, се нарича *покритие*.

Максимално по мощност съчетание се нарича съчетанието с максимален брой ребра (в графа G то може да не е единствено).

Максимално по тежест (тегло) съчетание се нарича съчетанието с максимално сумарно тегло на участващите в него ребра.

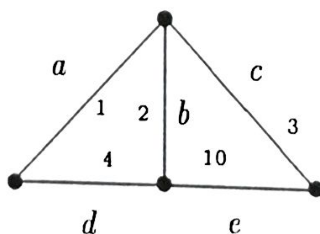
Минимално по мощност покритие е покритието с минимален брой ребра.

Минимално по тежест (тегло) покритие е покритието с минимално сумарно тегло на участващите в него ребра.

Очевидно:

1. Всяко подмножество на едно съчетание също е съчетание.
2. Всяко множество от ребра, включващо покритие, също е покритие.
3. Максималното по мощност съчетание не винаги е максимално по тегло.
4. Минималното по мощност покритие не винаги е с минимална тежест.

ПРИМЕР 8.1. В графа G, изобразен долу



1. Множествата ребра $\{a\}$, $\{a,\ e\}$ са сдвоявания, като второто от тези сдвоявания е максимално по мощност. Сдвояването $\{d,\ c\}$ също е максимално по мощност.

2. Множествата $\{a,\ e\}$, $\{d,\ c\}$, $\{a,\ b,\ c\}$ са покрития. Първите две от тези покрития са минимални по мощност.

3. Сдвояването $\{d,\ c\}$, което е с максимална мощност, е с по-малко тегло от сдвояването $\{e\}$ $(7 < 10)$.

4. Покритието $\{d,\ c\}$ е минимално по мощност, но неговото тегло е по-голямо от теглото на покритието $\{a,\ b,\ c\}$ $(7 > 6)$.

1. Оптимални сдвоявания в биполярни графи

Ще дадем алгоритми за намиране на максимални по мощност и максимални по тежест сдвоявания в биполярни графи $G = (X, Y, A)$. Специфичните свойства на биполярния граф дават възможност лесно, с използване на потокови алгоритми, да се намират цитираните горе сдвоявания.

МАКСИМАЛНО ПО МОЩНОСТ СДВОЯВАНЕ. *СТЪПКА 1.* Ориентирайте всички ребра в G от X към Y.

СТЪПКА 2. Въведете връх s (източник) и дъги (s, x), ориентирани от източника s към всеки връх $x \in X$.

СТЪПКА 3. Въведете връх t (сток) и дъги (y, t), ориентирани от всеки връх $y \in Y$ към стока t.

СТЪПКА 4. В графа G', получен след прилагането на горните три стъпки, на всяка дъга съпоставете капацитет (пропускателна способност) 1.

СТЪПКА 5. Приложете алгоритъма на Форд-Фалкерсон за намиране на поток с максимална величина в мрежата, дефинирана в *стъпка 4* (началният поток е нулев).

СТЪПКА 6. Ненулевите дъги на получения максимален поток задават максимално по мощнот сдвояване в G.

ОБОСНОВКА НА АЛГОРИТЪМА. 1. Очевидно във всяка дъга на графа G', потокът, протичащ по дъгата е или 0 или 1, тъй като всички капацитети са $= 1$.

2. Всеки поток в мрежата G' (следователно и максималния) индуцира сдвояване в графа G, тъй като от всеки връх на X излиза най-много една единица поток и във всеки връх на Y влиза най-много една единица поток.

3. На всяко сдвояване в G съответства поток в G'.

4. Да допуснем, че сдвояването в G, индуцирано от максималния поток в G', не е максимално по мощност. Тогава в G ще съществува "по-мощно" сдвояване и съответният на това сдвояване поток в G' ще се окаже "по-максимален" от максималния поток. Противоречие. Отхвърляме допускането.

Използването на потокови алгоритми със сложност $O(n^3)$ гарантира същата сложност за намиране на максимално сдвояване. Особените свойства на разглежданата мрежа обаче, дават възможност да се строят по-ефективни алгоритми. Такъв е например алгоритъмът на Хопкрофт-Карп от [49], чиято сложност е $O(^{5/2})$.

СЪВЪРШЕНИ СДВОЯВАНИЯ С МИНИМАЛНО ТЕГЛО. Формулираната в параграф 1.5 *задача за назначенията* се описва с пълния биполярен граф $K_{n,n} = (X, Y, A)$, в който $|X| = |Y| = n$. Очевидно и в този случай, както при максималните по мощност сдвоявания в биполярни графи, могат да се използват потоковите алгоритми. По-точно: 1. Добавя се изкуствен източник s с дъги (s, x_i), $x_i \in X$ с единичен капацитет и нулева цена (тегло).

2. Добавя се изкуствен сток (краен пункт) t и дъги (y_i, t), $y_i \in Y$ с единичен капацитет и нулева цена (тегло).

3. Ребрата се ориентират от X към Y.

4. Дъгите (x, y) са с единичен капацитет.

Тогава очевидно алгоритъмът за максимален поток с минимална стойност от s към t ще генерира в изходния граф $K_{n,n}$ съвършено съчетание с минимално сумарно тегло (цена). Припомняме, че едно съчетание M е съвършено, когато насища всички върхове на графа G, т.е. всеки връх от G е инцидентен с ребро от M.

Специалната структура и свойства на биполярните графи съкращават много стъпки на прилагания алгоритъм за намиране на поток с минимална цена.

МАКСИМАЛНО ПО ТЕЖЕСТ СДВОЯВАНЕ. С малка модификация, както в предишния случай (сдвояване с минимално тегло), може да се използва алгоритъма за намиране на поток от v единици с минимална стойност.

Ако цените (теглата) на ребрата в биполярния граф $G = (X, Y, A)$ са $p_i > 0$, можем да разгледаме графа G', в който:

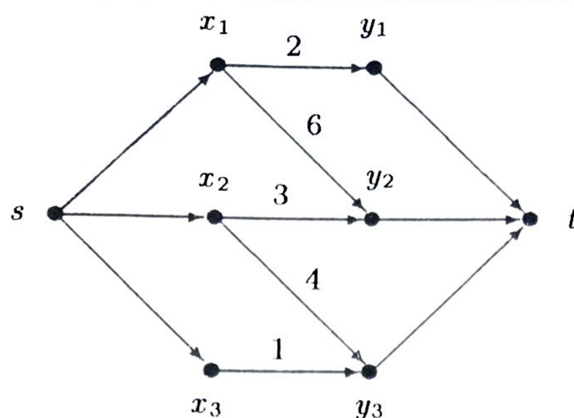
1. Добавени са изкуствен източник s и сток t.

2. Дъгите от вида (s, x) и (y, t) са с единичен капацитет и нулева цена.

3. Ребрата на G са ориентирани от X към Y.

4. Дъгите (x, y) са с единичен капацитет и тегло (цена) $\theta - p_i$, където θ е достатъчно голямо цяло число (напр. $\theta \geq \max\{p_i\}$. Напомняме, че (вж. параграф 2.5) когато всички капацитети са цели числа, то потокът в дъгите също ще бъде цяло число — в конкретния случай 0 или 1.

ПРИМЕР 8.2. Да разгледаме следния биполярен граф $G = (X, Y, A)$,



Черт. 2.6

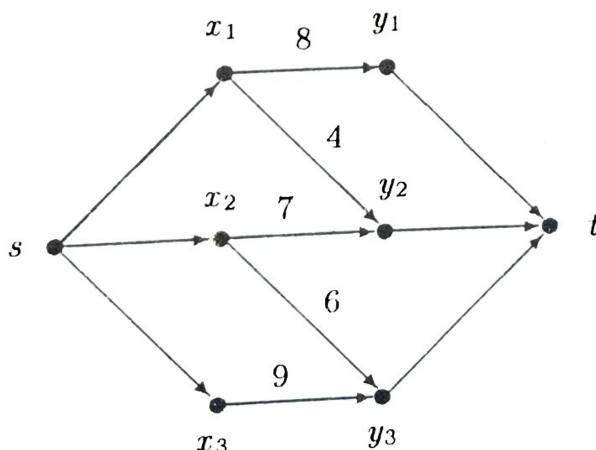
допълнен с изкуствен източник s и сток t, както беше предложено в теоретичните бележки. Числата, съпоставени на дъгите (x_i, y_i), са техните цени (тегла).

1. Очевидно алгоритъмът за поток с максимална величина (Форд-Фалкерсон) след приключване на работа ще намери поток по веригите

$$\begin{array}{ll} (s, x_1),\ (x_1, y_1),\ (y_1, t) & \text{— 1 единица} \\ (s, x_2),\ (x_2, y_2),\ (y_2, t) & \text{— 1 единица} \\ (s, x_3),\ (x_3, y_3),\ (y_3, t) & \text{— 1 единица} \end{array}$$

т.е. максималният поток ще бъде с величина 3. Този максимален поток, по-точно дъгите от G с ненулев поток представят съчетание в графа G с максимална мощност (максимален брой ребра).

2. Нека сега модифицираме теглата (цените) на ребрата от G като приемем, че θ е достатъчно голямо число, напр. $\theta = 10$, т.е. за всяко тегло p_i на ребро от G, модифицираното тегло е $\theta - p_i$.



Черт. 2.7

Очевидно, алгоритъмът за намиране на поток от v единици с минимална цена ще генерира веригите:

v	Вериги	Цена
1 ед.	$s,\ x_1,\ y_2,\ t$	4
2 ед.	$s,\ x_1,\ y_2,\ t$ и $s,\ x_2,\ y_3,\ t$	10
3 ед.	$s,\ x_1,\ y_1,\ t;\ s,\ x_2,\ y_2,\ t$ и $s,\ x_3,\ y_3,\ t$	24

Всяка такава верига (поток) генерира в изходния граф G съчетание от v на брой ребра, което е с максимално сумарно тегло измежду всички съчетания от v на брой ребра. В конкретния случай:

1. Потокът от 1 единица с минимална цена 4 е съответен на сдвояването $M_1 = \{(x_1,\ y_2)\}$, което е максималното по тежест едноелементно сдвояване в изходния граф (тегло 6).

2. Потокът от 2 единици с минимална сумарна цена 10 е съответен на двуелементното сдвояване $M_2 = \{(x_1,\ y_2),\ (x_2,\ y_3)\}$, което в изходния граф G е максималното по тежест двуелементно сдвояване (тегло 10).

3. Потокът от 3 единици с минимална сумарна цена 24 е съответен на триелементното сдвояване $M_3 = \{(x_1,\ y_1),\ (x_2,\ y_2),\ (x_3,\ y_3)\}$, което в изходния граф G е максималното по тежест триелементно сдвояване (тегло 6).

Максималното по тежест сдвояване в G се получава като

$$M_{\max} = \max_{\forall i}\{M_i\}.$$

Най-големият брой ребра на G, участващи в едно сдвояване M, съвпада с величината на максималния поток в модифицирания граф.

2. Унгарски алгоритъм

Ще дадем един ефективен алгоритъм, предложен от Кун (унгарски математик), за намиране на максимално по тежест сдвояване в биполярен граф $G = (X, Y, A)$.

Проблемът може да бъде формулиран като задача на линейното оптимиране по следния начин (вж. параграф 2.1):

$$\max \sum_i \sum_j c_{ij} x_{ij} \tag{8.1}$$

при ограничения

$$\begin{aligned} &\sum_j x_{ij} \le 1, && \text{за } \forall i \in X \\ &\sum_i x_{ij} \le 1, && \text{за } \forall j \in Y \\ &x_{ij} \ge 0, && \text{за } \forall i \in X,\ j \in Y. \end{aligned} \tag{8.2}$$

Очевидно този проблем е близък до транспортната задача и има целочислено решение. Ясно е, че $x_{ij} = 1$ тогава и само тогава, когато реброто (i, j), $i \in X$, $j \in Y$, участва в оптималното сдвояване.

За да бъде разбрана идеята на алгоритъма, ще припомним следното (вж. параграф 2.1): Задачите

$Max\ CX$	$Min\ BU$
$AX \leq B$ $X \geq 0$	$UA \geq C$ $U \geq 0$

се наричат дуални задачи. Обикновено първата от тях се нарича права, а втората — дуална на първата (и обратно).

Основният резултат, който получихме по-рано в линейното оптимиране е, че необходимо и достатъчно условие плановете (допустимите решения) X и U на две дуални задачи да бъдат оптимални, е да се удовлетворяват

$$U(AX - B) = UAX - UB = 0 \text{ и} \tag{8.3}$$

$$(C - UA)X = CX - UAX \leq 0. \tag{8.4}$$

Следователно $UB = UAX \geq CX$. С други думи стойността на дуалната целева функция (min) е винаги $\geq$ от стойността на целевата функция в правата задача (max).

Ако $UB = CX$, очевидно X и U ще бъдат оптимални решения на съответните задачи. Това е в сила, когато $(C - UA)X = 0$. Окончателно получаваме следните условия за оптималност

$$U(AX - B) = 0 \text{ и } (C - UA)X = 0. \tag{8.5}$$

Да формулираме дуалната задача на задачата (8.1) — (8.2)

$$\min \sum u_i + \sum v_j \tag{8.6}$$

при условия

$$(8.7) \qquad \begin{array}{ll} u_i + v_j \geq c_{ij}, & \text{за } \forall i,\ j; \\ u_i,\ v_j \geq 0, & \text{за } \forall i,\ j; \end{array}$$

където дуалните променливи u_i са съответни на върховете $i \in X$, а v_j на върховете $j \in Y$.

От направените коментари е ясно, че

а) ако $x_{ij} = 1$, то $u_i + v_j = c_{ij}$;

б) ако $u_i > 0$, то $\sum_j x_{ij} = 1$;

в) ако $v_j > 0$, то $\sum_i x_{ij} = 1$.

Идеята на *унгарския алгоритъм* е следната: Стартира се от допустими решения на правата и дуалната задача, така че да са удовлетворени условията а) и в), като алгоритъмът се опитва да намери увеличаващ (аугментален) път (вж. параграф 1.5 и теорема 5.12 — теорема на Берж) в подграфа, образуван от ребрата, за които $u_i + v_j = c_{ij}$.

Ако такъв път бъде намерен, новото сдвояване ще бъде допустимо и по-малко условия б) ще бъдат нарушени.

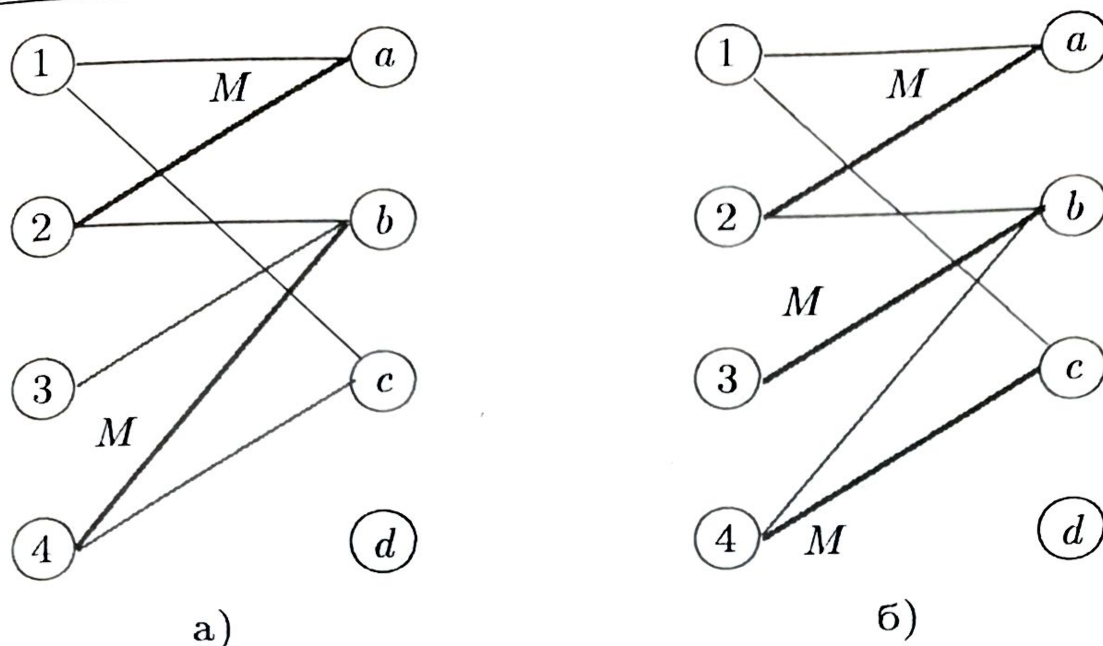
Ако няма такъв път, дуалните променливи се коригират, така че поне едно допълнително ребро да бъде добавено към подграфа.

Ще илюстрираме казаното като разгледаме следния пример от [47] за биполярен граф с матрица на теглата c_{ij}:

	a	b	c	d	u_i
1	32	18	32	26	32
2	22	24	12	16	22
3	24	30	26	24	28
4	26	30	28	20	28
v_j	0	2	0	0	

Дуално допустимо решение може да получите като например положите $v_1 = 0$ и поради $u_1 + v_1 = c_{11} \Rightarrow u_1 = 32$. От $u_2 + v_1 = c_{21} \Rightarrow u_2 = 22$ и т.н.

В горната таблица е показано едно възможно решение на дуалната задача. Едно възможно решение на правата задача е сдвояването $M = \{(2, a),\ (4, b)\}$, т.е. $x_{2a} = x_{4b} = 1$.



Черт. 2.8

Обърнете внимание, че условията а) и в) са изпълнени, докато условието б) не е. На черт. 2.8 а) са изобразени всички ребра на биполярния граф, за които е изпълнено условието $u_i + v_j = c_{ij}$.

Очевидно съществува аугментална верига за сдвояването M — веригата 3, b, 4, c. Това е верига, на която началният и крайният връх са ненаситени и след превръщането на тъмните ребра (ребрата участващи в сдвояването) в светли (неучастващи в M) и обратно — преобразуването на светлите ребра в тъмни, ще достигнем до ново сдвояване, както е показано на черт. 2.8 б).

По този начин получихме едно ново допустимо решение

$$x_{2a} = x_{3b} = x_{4c} = 1$$

на правата задача, без да се е променило решението на дуалната задача. Условията а) и в) остават удовлетворени, като при това условието б) съответно за връх 3 сега е удовлетворено.

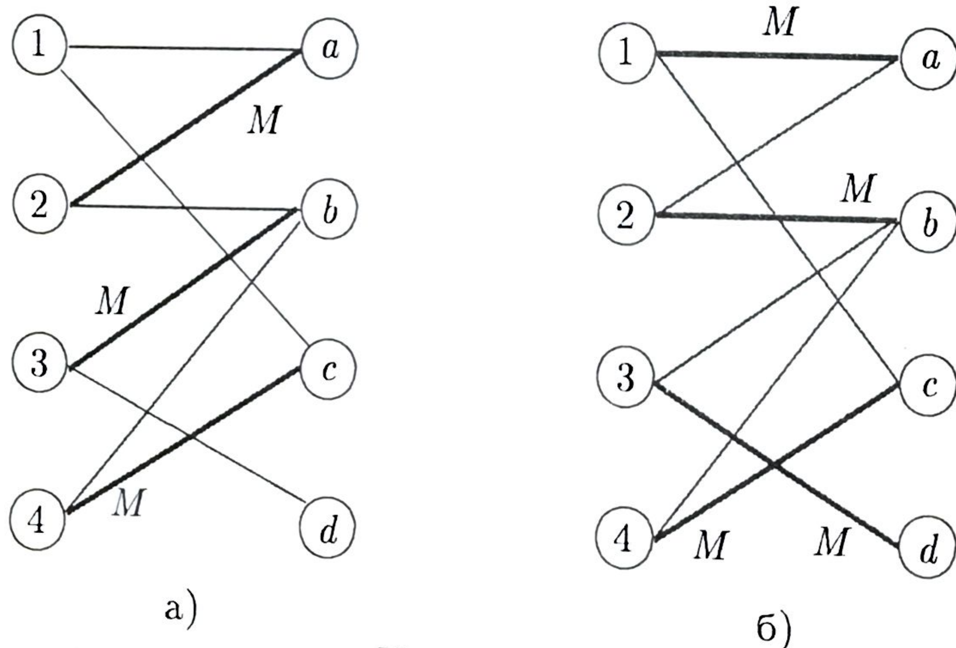
В графа от черт. 2.8 б) не могат да бъдат намерени увеличаващи (аугментални) вериги. При тази ситуация унгарският алгоритъм коригира стойностите на дуалните променливи. Той намалява с δ, напр. $\delta = 4$, всяко u_i и добавя $\delta = 4$ към всяко

$v_j = 1,\ 2,\ 3$, т.е. получава се следната таблица:

	a	b	c	d	u_i
1	32	18	32	26	28
2	22	24	12	16	18
3	24	30	26	24	24
4	26	30	28	20	24
v_j	4	6	4	0	

в която е дадено новото решение на дуалната задача.

Тъй като с $\delta = 4$ са намалени дуалните променливи съответни на върховете в X и с $\delta = 4$ са увеличени съответните дуални променливи на върховете в Y, които са били свързани с ребро (вж. черт. 2.8 б)), т.е. върховете, за които $u_i + v_j = c_{ij}$, това свойство ще продължава да е налице за тези върхове. Нещо повече, промяната на дуалните променливи води до появата на ребро $(3, d)$, за което $u_3 + v_d = c_{3d}$, т.е. реброто $(3, d)$ се включва в подграфа, както е показано на черт. 2.9 а):



Черт. 2.9

От черт. 2.9 а) се вижда съществуването на увеличаваща верига d, 3, b, 2, a, 1, която води до ново решение, показано на черт. 2.9 б).

За да може да се намират аугментални вериги и да се представя по подходящ (за компютъра) начин промяната на дуалните променливи, в алгоритъма се използва маркиране на върховете в графа. Ще дадем едно по-формално описание на този алгоритъм.

УНГАРСКИ АЛГОРИТЪМ ЗА МАКСИМАЛНО ПО ТЕЖЕСТ СДВОЯВАНЕ. *Инициализация:* Нека $u_i = \max_j\{c_{ij}\}$, $i \in X$; $v_j = 0$, $\pi_j = \infty$, $j \in Y$. Конструира се подграф, състоящ се от ребрата, за които $u_i + v_j = c_{ij}$. За $\forall i \in X$ се избира първото ребро (i, j), такова, че j не е съчетано и това ребро се включва в първоначалното сдвояване M. Всички върхове на графа са неразглеждани и немаркирани.

СТЪПКА 1. Маркират се всички ненаситени върхове (върховете неинцидентни с ребра от M) $i \in X$ с $p(i) = 0$.

СТЪПКА 2. Избира се произволен неразгледан, но маркиран връх $i \in X$, или $j \in Y$, за който $\pi_j = 0$. Ако такъв няма, се преминава на *стъпка 5*.

СТЪПКА 3. Ако върхът, избран в *стъпка 2* е $i \in X$, тогава за всяко ребро (i, j), невключено в M, върхът $j \in Y$ се маркира с $p(j) = i$ ако $u_i + v_j - c_{ij} < \pi_j$ и се заменя π_j с $u_i + v_j - c_{ij}$.

Ако върхът, избран в *стъпка 2* е $j \in Y$, се установява дали j е ненаситен връх. Ако това е така, се отива на *стъпка 4*. В противен случай съществува ребро (i, j) в M. Маркира се върхът $i \in X$ с $p(i) = j$.

Във всеки от случаите ($i \in X$ или $j \in Y$) се връщаме на *стъпка 2*.

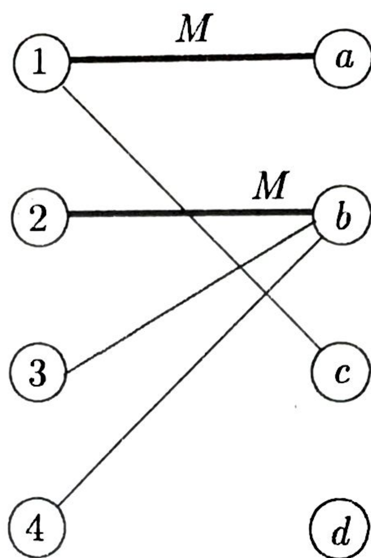
СТЪПКА 4. Съществува (намерена е) увеличаваща верига с край във върха $i \in X$ или $j \in Y$. Трасираме тази верига като се използва функцията $p(\ )$. Увеличаваме съчетанието като включваме към M всички ребра, невключени в текущото M и отстраняваме всички ребра, които са в текущото M. За всяко $j \in Y$ полагаме $\pi_j = \infty$. Изтриваме всички маркировки и се връщаме на *стъпка 1*.

СТЪПКА 5. Изчисляваме $\delta_1 = \min\{u_i,\ i \in X\}$, $\delta_2 = \min\{\pi_j > 0,\ j \in Y\}$ и $\delta = \min\{\delta_1,\ \delta_2\}$. Извършваме присвояванията: $u_i = u_i - \delta$ за всеки маркиран връх $i \in X$; $v_j = v_j + \delta$ за всяко $j \in Y$ с $\pi_j = 0$. Полагаме $\pi_j = \pi_j - \delta$ за всеки маркиран връх $j \in Y$ с $\pi_j > 0$. Ако $\delta = \delta_2$, се връщаме на *стъпка 2*. В противен случай сдвояването с максимално тегло е намерено.

Да се върнем отново на разгледания пример. Да означим множеството от неразгледани и маркирани върхове с L. Първоначалното множество от дуални променливи и първоначалното сдвояване са следните:

	a	b	c	d	u_i
1	32	18	32	26	32
2	22	24	12	16	24
3	24	30	26	24	30
4	26	30	28	20	30
v_j	0	0	0	0	
π_j	∞	∞	∞	∞	

,



СТЪПКА 1. Върховете 3 и 4 са ненаситени. Маркираме ги с $p(3) = 0$ и $p(4) = 0$.

СТЪПКА 2. $L = \{3,\ 4\}$. Избираме $i = 3$.

СТЪПКА 3. Маркираме върховете $j \in Y$, т.е. $p(a) = 3$, $p(b) = 3$, $p(c) = 3$, $p(d) = 3$. Новият π-вектор е $(6, 0, 4, 6)$.

СТЪПКА 2. $L = \{4,\ b\}$. Избираме $i = 4$.

СТЪПКА 3. Маркираме $j \in Y$, т.е. $p(a) = 4$, $p(c) = 4$. Векторът $\pi = (4, 0, 2, 6)$.

СТЪПКА 2. $L = \{b\}$. Избираме $j = b$.

СТЪПКА 3. Върхът b е наситен. Маркираме върха 2 с $p(2) = b$.

СТЪПКА 2. $L = \{2\}$. Избираме $i = 2$.

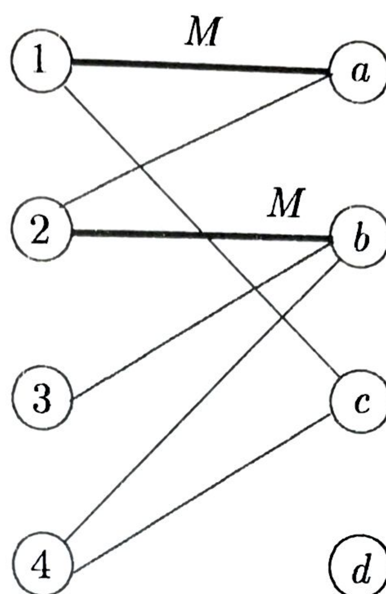
СТЪПКА 3. Маркираме $j \in Y$, т.е. $p(a) = 2$. Векторът $\pi = (2, 0, 2, 6)$.

СТЪПКА 2. Преминаваме към *стъпка 5.*

СТЪПКА 5. $\delta_1 = \min\{32,\ 24,\ 30,\ 30\} = 24$; $\delta_2 = \min\{2,\ 2,\ 6\} = 2$; $\delta = 2$. Върховете 2, 3 и 4 са маркирани. Намаляваме с δ дуалните стойности на тези върхове. Добавяме δ към v_b. Вадим δ от π_a, π_c и π_d. Получаваме следния резултат:

	a	b	c	d	u_i
1	32	18	32	26	32
2	22	24	12	16	22
3	24	30	26	24	28
4	26	30	28	20	28
v_j	0	2	0	0	
π_j	2	0	0	4	

,



СТЪПКА 2. $L = \{c\}$. Избираме $j = c$.

СТЪПКА 3. Върхът c е ненаситен.Преминаваме към *стъпка 4*.

СТЪПКА 4. Увеличаваме сдвояването като се връщаме назад: $p(c) = 4$, $p(4) = 0$. Векторът $\pi = (\infty, \infty, \infty, \infty)$. Сдвояването $M = \{(1, a), (2, b), (4, c)\}$.

СТЪПКА 1. Върхът 3 е ненаситен. Маркираме го с $p(3) = 0$.

СТЪПКА 2. $L = \{3\}$. Избираме $i = 3$.

СТЪПКА 3. Маркираме $p(a) = 3$, $p(b) = 3$, $p(c) = 3$, $p(d) = 3$. Векторът $\pi = (4, 0, 2, 4)$.

СТЪПКА 2. $L = \{b\}$. Избираме $j = b$.

СТЪПКА 3. Върхът b е наситен. Маркираме $p(2) = b$.

СТЪПКА 2. $L = \{2\}$. Избираме $i = 2$.

СТЪПКА 3. Маркираме $p(a) = 2$. Векторът $\pi = (0, 0, 2, 4)$.

СТЪПКА 2. $L = \{a\}$. Избираме $j = a$.

СТЪПКА 3. Върхът a е наситен. Маркираме $p(1) = a$.

СТЪПКА 2. $L = \{1\}$. Избираме $i = 1$.

СТЪПКА 3. Маркираме $p(c) = 1$. Векторът $\pi = (0, 0, 0, 4)$.

СТЪПКА 2. $L = \{c\}$. Избираме $j = c$.

СТЪПКА 3. Върхът c е наситен. Маркираме $p(4) = c$.

СТЪПКА 2. $L = \{4\}$. Избираме $i = 4$.

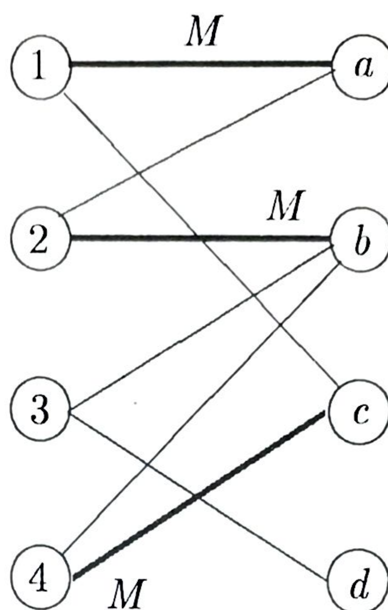
СТЪПКА 3. Няма нови маркировки.

СТЪПКА 2. Преминаваме на *стъпка 5*.

СТЪПКА 5. $\delta_1 = \min\{32,\ 22,\ 28,\ 28\} = 22$; $\delta_2 = \min\{4\} = 4$; $\delta = 4$. Върховете 1, 2, 3 и 4 са маркирани. Изваждаме δ от

дуалните стойности на тези върхове и добавяме δ към v_a и v_2. Изваждаме δ от π_d. Полученият резултат е следния:

	a	b	c	d	u_i
1	32	18	32	26	28
2	22	24	12	16	18
3	24	30	26	24	24
4	26	30	28	20	24
v_j	4	6	4	0	
π_j	0	0	0	0	



СТЪПКА 2. $L = \{d\}$. Избираме $j = d$.

СТЪПКА 3. Върхът d е ненаситен. Преминаваме към *стъпка 4.*

СТЪПКА 4. Увеличаваме сдвояването. $M = \{(1,a), (2,b), (3,d), (4,c)\}$. Векторът $\pi = (\infty,\infty,\infty,\infty)$.

СТЪПКА 1. Няма върхове $i \in X$, които са ненаситени (всеки от върховете 1, 2, 3, 4 участва в сдвояването M).

СТЪПКА 2. Преминаваме към *стъпка 5.*

СТЪПКА 5. $\delta = \delta_1$. Край.

Унгарският алгоритъм има сложност $O(n^3)$.

Припомняме, че сложността на модификацията на Едмондс и Карп за алгоритъма на Форд-Фалкерсон (максимален поток) е $O(m^2.n)$.

ЗАДАЧА 8.1. Да се определи сложността на предложения в началото на параграфа алгоритъм за определяне максимално по тежест сдвояване, използващ алгоритъма за поток с минимална стойност.

3. Оптимални сдвоявания в произволни графи

МАКСИМАЛНО ПО МОЩНОСТ СДВОЯВАНЕ. Когато графът G е произволен, той може да не е биполярен, т.е. в него може да съществува цикъл с нечетна дължина (вж. теорема 2.1 в глава 1). Евентуалното наличие на нечетни цикли поражда трудности и прави невъзможно използването на потоковите алгоритми както в случая на биполярен граф.

Проблемът за намиране на максимално по мощност сдвояване в произволен граф може да се формулира като задача на линейното оптимиране:

$$\sum_{(i,j)} x(i,j) \longrightarrow \max$$

при ограничения

$$\sum_{j}(x(j,i) + x(i,j)) \leq 1, \quad \text{за } \forall i;$$
$$0 \leq x(i,j) \leq 1, \quad \text{за } \forall \text{ ребро } (i,j),$$

където $x(i,j)$ определя броя на включените в сдвояването ребра (i,j).

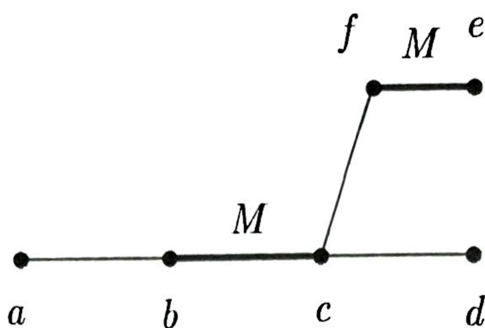
Очевидно всяко сдвояване удовлетворява ограниченията на тази задача на линейното оптимиране, ако положим $x(i,j) = 1$ за ребрата (i,j), влизащи в сдвояването. За съжаление обаче, ограниченията на тази задача могат да се удовлетворяват и от нецелочислени стойности на променливите $x(i,j)$. С други думи решението на тази задача на линейното оптимиране не винаги задава сдвояване. С редица допълнителни ограничения този проблем може да бъде решен, но това води до задача на линейното оптимиране с твърде големи размери и до неефективност на алгоритъма. Ето защо ще разгледаме алгоритъма, предложен от Едмондс в [46].

Основната идея на алгоритъма на Едмондс се базира на теорема 5.12 (теорема на Берж) от параграф 1.5. Според тази теорема сдвояването M в произволен граф G е максимално тогава и само тогава, когато в графа не съществувт усилващи относно M вериги.

Ще въведем необходимите за целта понятия и терминология.

1. Ако M е сдвояване в графа G, *алтернативна (редуваща се)* верига относно M ще наричаме всяка проста верига, в която

от всеки две съседни ребра едното принадлежи, а другото не принадлежи на сдвояването M. Например



веригата a, b, c, f, e е алтернативна верига.

2. Ребрата, включени в сдвояването M ще наричаме *тъмни ребра*, а невключените в M — *светли ребра*. Ребрата (b,c) и (f,e) от горния пример са тъмни, а останалите — светли.

3. Върховете, инцидентни с ребро от M ще наричаме *наситени (тъмни)*, а останалите — *ненаситени (светли, експонирани)* върхове.

4. Алтернативната верига се нарича *увеличаваща верига (усилваща, аугментална)*, ако крайните ѝ върхове са ненаситени. В примера, който разгледахме, веригата a, b, c, d е алтернавтивна верига с експонирани крайни върхове a и d. Тази верига е аугментална, защото светлите ребра в нея — (a,b) и (c,d) могат да се преобразуват в тъмни, а тъмното ребро (b,c) — в светло, което ще генерира ново сдвояване с по-голяма мощност.

5. *Алтернативно дърво* относно сдвояването M се нарича дървото T, за което:

а) един връх, наречен корен на T, се явява експониран (ненаситен);

б) всички започващи от корена вериги са алтернативни;

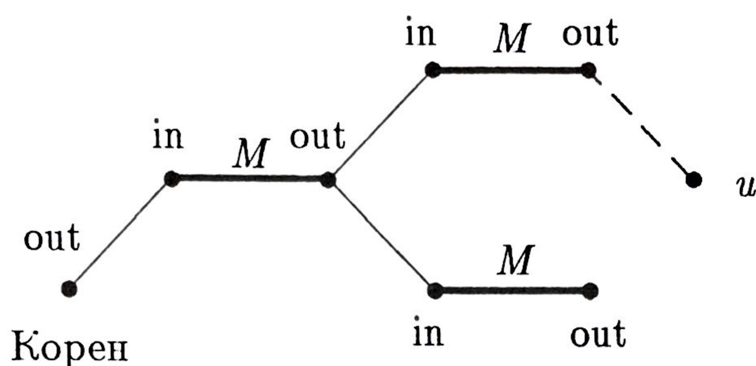
в) всички максимални вериги, започващи от корена на T, съдържат четно число ребра.

В алтернативното дърво всяка верига, започваща от корена, е алтернативна и нейното първо ребро не принадлежи на сдвояването M, тъй като коренът е експониран връх.

Да разбием (маркираме) върховете на дървото на два класа — *вътрешни* и *външни върхове*. Първите ще бележим (маркираме) със символа "in", а вторите — със символа "out". Коренът на дървото считаме винаги за "out", т.е. външен връх. Върховете по продължение на всяка верига, започваща от корена, последователно и алтернативно се маркират с "in" и "out". Например

Алтернативно дърво T

(пунктираното ребро $\notin T$)



Ребра от дървото $\in M$

Ребра от дървото $\notin M$

Други ребра на графа

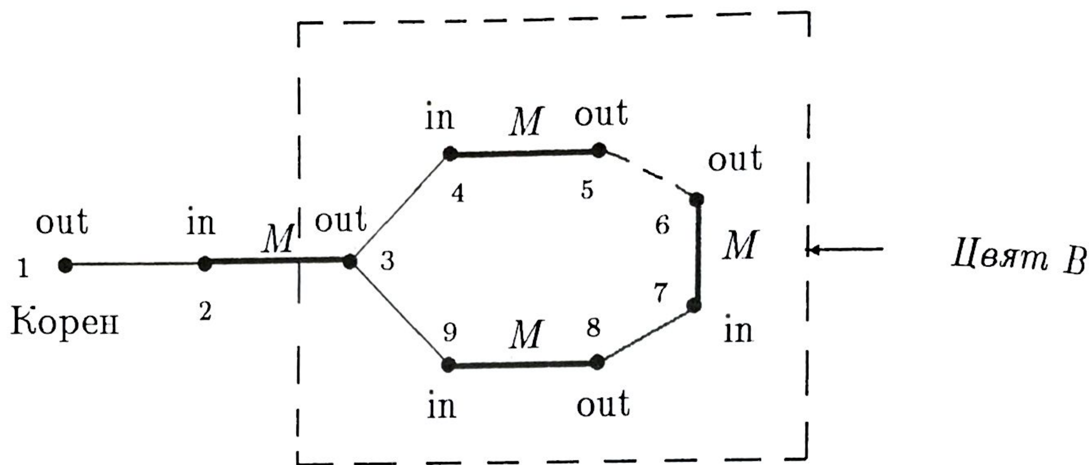
Очевидно всеки "in"-връх е от степен 2, а "out"-върховете са с произволни степени.

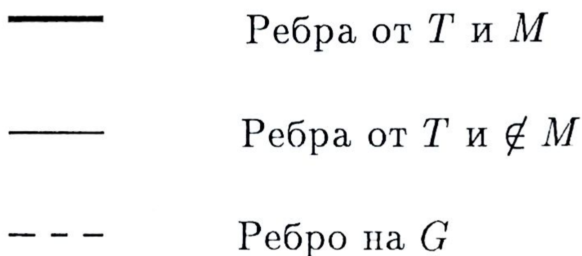
6. *Аугментално дърво* — това е алтернативно дърво (относно сдвояването M), в което съществува ребро от някой out-връх v до експониран връх u (непринадлежащ на дървото).

Очевидно единствената верига от корена до върха v, допълнена с реброто (v, u) ще бъде аугментална. В горния пример такава е веригата от корена до върха u.

7. *Цвят (букет)* относно сдвояването M се нарича аугментална верига, на която началният и крайният експонирани върхове съвпадат, т.е. цикъл, тъй като броят участващи във веригата ребра е нечетен. Използваният термин идва от англ. *blossom*.

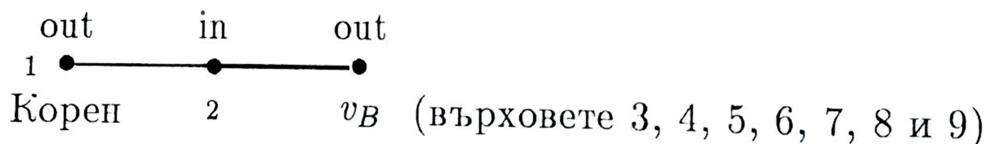
На черт. 2.10 е илюстрирано понятието цвят (букет):





Черт. 2.10

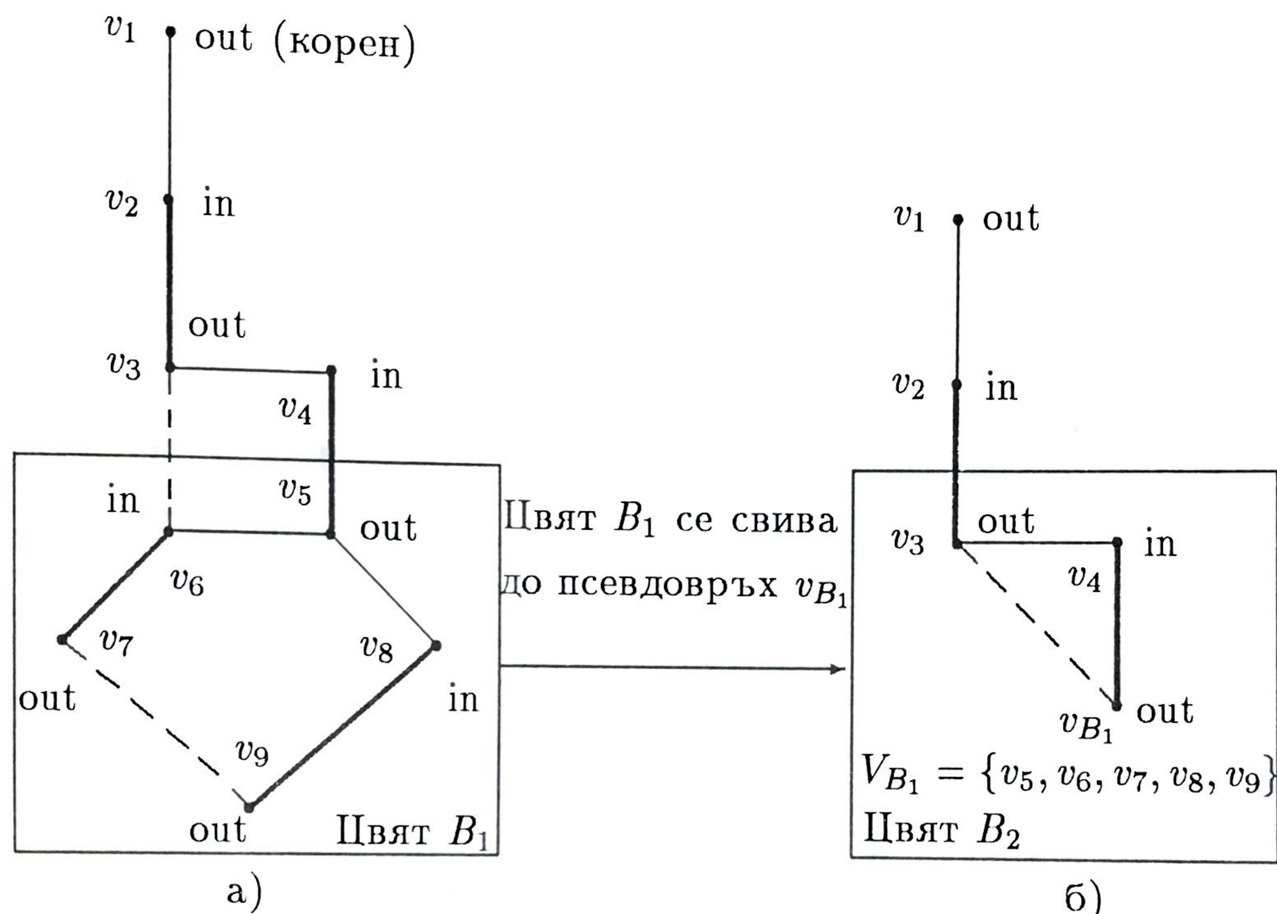
8. В процеса на работа на алгоритъма на Едмондс цветовете се *свиват* с цел да се получи по-прост граф. По-точно, *свиването на един цвят* B се състои в замяна на върховете от цикъла (цвета B) с нов псевдовръх v_B. При това реброто (v_B, u) се добавя винаги, когато съществува ребро от връх на цикъла до друг връх u, непринадлежащ на цикъла. Свиването на цвят B от черт. 2.10 води до



В получения след свиването граф е възможно върхът v_B и други върхове или псевдовърхове (съответни на по-рано свити цветове) да образуват нов цвят. Този нов цвят отново се свива и т.н. Последният цвят B_k, който не се съдържа в други цветове, се нарича *краен цвят*.

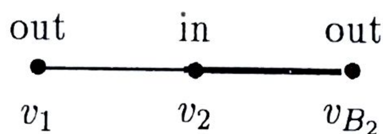
Да разгледаме черт. 2.11 а). Обръщаме внимание, че на чертежа не е изобразен целия граф, а част от него и по-точно дадено е алтернативно дърво построено в графа, относно сдвояването M (ако това беше целия граф G, сдвояването очевидно

щеше да е максимално поради наличието само на един експониран връх v_1).



Черт. 2.11

Алтернативно дърво след свиване на цветовете:



$$V_{B_2} = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$$

По-късно ще станат ясни основанията за този процес на свиване на цветовете и получаването на максимално M. В алгоритъма след свиването започва обратен процес на "разцъвтяване", т.е. замяна на псевдовърховете с истинските върхове, образували цвета (цикъла).

Когато един цвят B е свит, съответният му псевдовръх v_B се счита за out-връх, за да може структурата на оставащото алтернативно дърво да бъде коректна, т.е. след свиването алтернативното дърво да се запазва в получавания граф.

Забележка: Обърнете внимание, че всеки връх от един цвят B може да бъде маркиран с in или out. Например връх 4 от черт. 2.10, който е маркиран с in (веригата 1, 2, 3, 4), можеше да бъде маркиран с out, ако маркирането се осъществи чрез веригата 1, 2, 3, 9, 8, 7, 6, 5, 4 (дължините на двете вериги са с различна четност).

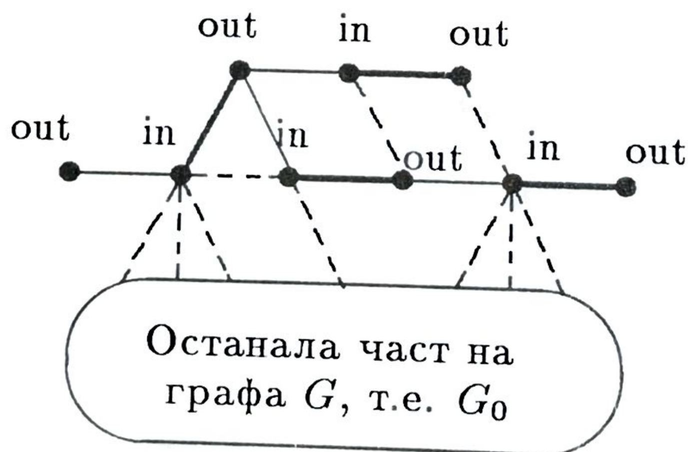
Лесно се доказва следната:

▷ ТЕОРЕМА 8.1. *Ако B е цвят с нечетен брой върхове V_B и v е произволен връх от V_B, то в породения подграф $\langle V_B \rangle$ съществува максимално сдвояване, за което върхът v е експониран.* ◁

9. Ще дефинираме още едно важно понятие, свързано с алгоритъма за намиране на максимално по мощност сдвояване в произволен граф.

Унгарско дърво (hungarian tree) — това е алтернативно дърво в графа, такова, че всяко ребро на графа, на което единият край е out-връх в дървото, другият му край е in-връх в дървото.

На черт. 2.12 е изобразено унгарско дърво.



Черт. 2.12

Ще докажем сега една много важна теорема, свързана с унгарските дървета, на която се базира идеята на алгоритъма за търсене на максимално сдвояване. Този резултат е получен от Едмондс в [46].

▷ ТЕОРЕМА 8.2. *Нека H е унгарско дърво в графа $G = (X, A)$ и $G_0 = \langle X - X_H \rangle$ е породеният подграф на G, където X_H са върховете на дървото H. Ако M_H е сдвояване в дървото H и $M^*_{G_0}$ е максималното сдвояване в G_0, то ребрата на множеството $M_H \cup M^*_{G_0}$ са максимално по мощност сдвояване в G.*

Коментар: Като следствие от тази теорема е ясно, че след като алгоритъмът намери унгарско дърво в графа G, той може да продължи да търси максимално сдвояване само в графа G_0, т.е. в останалата част на графа G, получена след отстраняване на върховете от дървото и инцидентните с тях ребра.

Доказателство: Нека множеството A от ребра на графа G е разбито на три подмножества:

$$\begin{aligned} A_H &= \{(x_i, x_j) | (x_i, x_j) \in A \text{ и } x_i,\ x_j \in X_H\}, \\ A_{HG_0} &= \{(x_i, x_j) | (x_i, x_j) \in A,\ x_i \in X_H \text{ и } x_j \in X - X_H\}, \\ A_{G_0} &= \{(x_i, x_j) | (x_i, x_j) \in A \text{ и } x_i,\ x_j \in X - X_H\}, \end{aligned}$$

т.е. A_H е множеството от всички ребра "принадлежащи" само на дървото, A_{HG_0} е множеството ребра, на които единият край е в дървото, а другият — в останалата част на графа и накрая A_{G_0} са ребрата на подграфа G_0.

Нека S е произволно сдвояване в G. По аналогия S може да се разбие по следния начин:

$$S = S_H \cup S_{HG_0} \cup S_{G_0},$$

където $S_H = S \cap A_H$, $S_{HG_0} = S \cap A_{HG_0}$ и $S_{G_0} = S \cap A_{G_0}$.

Тъй като $M^*_{G_0}$ е максималното сдвояване, то

$$|M^*_{G_0}| \geq |S_{G_0}|. \tag{8.8}$$

Да означим с G' графа, породен от ребрата $A_H \cup A_{HG_0}$.

Всички върхове $x_K \in X - X_H$, явяващи се крайни за ребра от A_{HG_0}, ще бъдат експонирани (ненаситени) относно M_H (припомняме, че M_H е сдвояването в унгарското дърво). Алтернативна верига, започваща от експониран връх $x_K \in X - X_H$ (или стартираща от корена на дървото H, който също е експониран), ще бъде аугментална, само когато тази верига завършва в друг връх $x_K \in X - X_H$.

Но аугменталната верига има нечетно число ребра, следователно, ако първото ребро на тази верига "идва" от експониран връх в in-връх, накрая веригата трябва от out-връх да завърши в експониран. Но дървото H е унгарско, следователно всичките ребра в A_{HG_0} свързват in-върхове на H с върхове на $X - X_H$, т.е. в графа G' няма аугментални вериги, т.е. M_H се явява максималното по мощност съчетание в G'. Тогава

$$|M_H| \geq |S_H| + |S_{HG_0}|. \tag{8.9}$$

От неравенствата (8.8) и (8.9) получаваме

$$(8.10) \qquad |M_H \cup M^*_{G_0}| \geq |S_H \cup S_{HG_0} \cup S_{G_0}| \geq |S|.$$

Тъй като S е напълно произволно сдвояване в G, от (8.10) следва,че $M_H \cup M^*_{G_0}$ е максималното по мощност сдвояване в G. ◁

10. Алтернативното дърво има за корен експониран връх и се строи чрез последователно и алтернативно добавяне на ребра, които не принадлежат - принадлежат на сдвояването M.

Строенето на алтернативното дърво се извършва докато:

(*) дървото стане аугментално; или

(**) на дървото се появява цвят (нечетен цикъл); или

(***) дървото стане унгарско.

В случай (*) броят на ребрата в сдвояването M може да се увеличи с единица, като се движим по аугменталната верига към корена на дървото, заменяйки светлите ребра с тъмни и обратно. След това, относно новополученото сдвояване, алгоритъмът, започвайки отначало (всички маркировки се игнорират), строи ново дърво (ако такова съществува) с корен — експониран връх.

В случай (**) полученият цвят се свива, както вече обяснихме в 8., и продължаваме да строим дървото.

От гледна точка на програмното реализиране на алгоритъма, свиването трябва да се разбира като маркиране на всички върхове от цвета като out-върхове и да се съхрани информацията, че те принадлежат на този цвят. Важно е да се помни и реда на свиване на цветовете, тъй като накрая цветовете трябва да "разцъфнат" в обратен ред.

В случая (***) върховете на унгарското дърво и инцидентните с тях ребра се отстраняват от графа и съгласно теорема 8.2, алгоритъмът се прилага са останалата част на графа.

АЛГОРИТЪМ НА ЕДМОНДС. *СТЪПКА 1.* Ако в графа G съществуват поне 2 ненаситени (експонирани) върха, изберете един такъв връх за корен. Маркирайте го с "out" и преминете към *стъпка 2.* В противен случай преминете към *стъпка 7.*

СТЪПКА 2. Изберете out-връх x от дървото. За всяко ребро (x, y):

а) ако y е експониран — преминете към *стъпка 3*;

б) ако y е in-връх — преминете към *стъпка 6*;

в) ако y не е от дървото и е наситен — преминете към *стъпка 4*.

СТЪПКА 3. Намерена е аугментална верига от корена до връх y. Постройте новото по-мощно сдвояване. Игнорирайте текущото дърво и всички маркировки,и се върнете на *стъпка 1.*

СТЪПКА 4. Добавете към алтернативното дърво реброто (x, y) и маркирайте връх y като вътрешен, т.е. с "in". Намерете реброто (y, z), принадлежащо на текущото сдвояване M, добавете го към дървото и маркирайте връх Z с "out".

Ако съществува ребро между z и друг out-връх, преминете към *стъпка 5.* В противен случай — към *стъпка 2.*

СТЪПКА 5. Получен е цвят. Свийте този цвят до псевдовръх и маркирайте псевдовърха с "out". Съхранете информация за поредността на свиването и преминете към *стъпка 2.*

СТЪПКА 6. Връщайте се към *стъпка 2* дотогава, докато (***) е единственият възникващ случай (унгарско дърво). Отстранете върховете на унгарското дърво и инцидентните с тях ребра от графа (вж. теорема 8.2). Полученият подграф считайте за граф G и се върнете на *стъпка 1.*

СТЪПКА 7. Намерете в последния граф G и във всяко отстранено унгарско дърво оптималното сдвояване $M_{\max}$ по следния начин. Разгънете ("разцъвтете") крайния цвят B_k, т.е. последния псевдовръх. Изберете в него онова сдвояване, спрямо което върхът x, който се сдвоява с още неразпуснат цвят, остава разекспониран (вж. теорема 8.1). Продължете процеса на "разпускане" на цветовете в ред, обратен на установения в *стъпка 5*, т.е. в ред, обратен на свиването, докато "разцъфнат" всички свити цветове и се получи максималното по мощност сдвояване в изходния граф G.

В [50] е даден алгоритъм за намиране на максимално по мощност сдвояване в произволен граф, чиято сложност е $O(n^{5/2})$, т.е. със същата сложност, с каквато е съответният алгоритъм на Хопкрофт-Карп при биполярни графи.

МАКСИМАЛНО ПО ТЕЖЕСТ СДВОЯВАНЕ. Ще разгледаме алгоритъм за намиране на сдвояване с максимално тегло в произволен граф $G = (X, A)$. Алгоритъмът е предложен от Едмондс и Джонсън в [51], като ние ще се придържаме към описанието в [47].

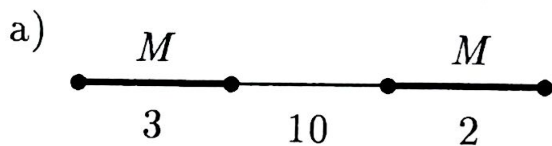
Този алгоритъм, както и предишния разгледан алгоритъм, строи алтернативно дърво. И в този случай се оказва, че сдвояването M е максмално тогава и само тогава, когато за него не

съществува аугментална верига. В този случай обаче, аугменталната верига се дефинира малко по-различно, като аугментацията е свързана не с броя на дъгите, а с тяхното сумарно тегло.

Аугментална верига — това е алтернативна верига, в която сумарното тегло на ребрата, неучастващи в сдвояването, е по-голямо от сумарното тегло на ребрата, участващи в сдвояването.

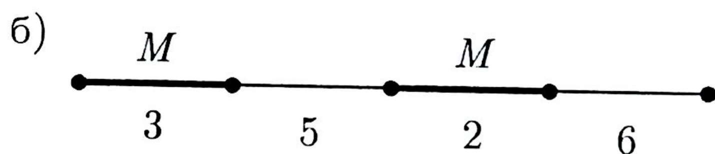
Ясно е, че ако в такава верига светлите ребра направим тъмни и обратно, ще получим ново сдвояване M с по-голямо тегло. Възможни са следните три вида увеличаващи вериги:

а)



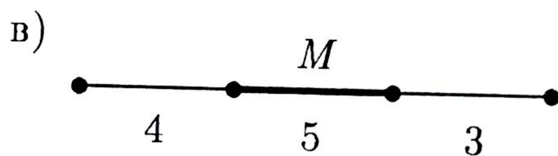
слаба увеличаваща верига

б)



неутрална увеличаваща верига

в)



силна увеличаваща верига

В горните три вериги сумарното тегло на ребрата, участващи в M е по-малко от сумарното тегло на ребрата, неучастващи в M, поради което веригите са аугментални. Очевидно, ако в тези вериги ребрата от M изключим от сдвояването, а непринадлежащите — включим в M, ще получим сдвояване с по-голямо тегло.

Веригите от тип а) се наричат слаби, защото броят на ребрата, непринадлежащи на M, е по-малък от броя на принадлежащите на M ребра. С други думи, новото сдвояване е с по-голяма тежест, а с по-малко ребра. По аналогични и естествени причини са избрани наименованията на веригите от тип б) и в).

И в този алгоритъм съществено се използва основния резултат в линейното оптимиране, а именно връзката между решенията на правата и дуалната задача (вж. параграф 2.1).

Нека $V = \{V_1,\ V_2,\ \ldots,\ V_z\}$ е множеството от всички подмножества от върхове на графа, включващи нечетен брой върхове. Нека T_m е множеството на всички ребра, двата крайни върха на които са от V_m. Да означим с T множеството $T = \{T_1,\ T_2,\ \ldots,\ T_z\}$, а с $2_{n_m} + 1$ — броя на върховете във V_m.

Ясно е, че никое сдвояване не съдържа повече от n_m ребра, принадлежащи на T_m.

Ако $a(i,j)$ е теглото на реброто (i,j) и $x(i,j) = 1$, когато реброто принадлежи на сдвояването ($x(i,j) = 0$, когато не принадлежи), задачата за максимално по тежест сдвояване може да се формулира като задача па линейното оптимиране така:

$$\sum_{(i,j)} a(i,j)x(i,j) \longrightarrow \max \tag{8.11}$$

при ограничения

$$\sum_{j} [x(i,j) + x(j,i)] \leq 1, \quad \text{за } \forall i \in X, \tag{8.12}$$

$$\sum_{i,j \in T_m} x(i,j) \leq n_m,\ m = 1,\ 2,\ \ldots,\ z, \tag{8.13}$$

$$x(i,j) \geq 0, \text{ за всички ребра } (i,j) \tag{8.14}$$

(реброто, съединяващо върховете i и j се обозначава чрез (i,j) или (j,i).)

Тъй като решаването на тази задача със симплекс-метода е неефективно, се постъпва по следния начин. Разглежда се дуалната на тази задача, а именно

$$\sum_{i \in X} u_i + \sum_{m=1}^{z} n_m v_m \tag{8.15}$$

при ограничения

$$u_i + u_j + \sum_{m:(i,j) \in T_m} v_m \geq a(i,j), \text{ за } \forall (i,j), \tag{8.16}$$

(8.17) $$u_i \geq 0, \text{ за } \forall i \in X,$$

(8.18) $$v_m \geq 0, \; m = 1, 2, ..., z.$$

Двойнствената променлива u_i е съответна на ограничението (8.12) за върха i, а двойнствената променлива v_m — на ограничението (8.13) за T_m.

Условията за равновесие на правата и дуалната задача, които получихме в параграф 2.1, в случая са следните

(8.19) $$x(i,j) > 0 \Longrightarrow u_i + u_j + \sum_{m:(i,j) \in T_m} v_m = a(i,j) \text{ за } \forall(i,j)$$

(8.20) $$u_i > 0 \Longrightarrow \sum_{j \in X} [x(i,j) + x(j,i)] = 1, \text{ за } \forall i \in X,$$

(8.21) $$v_m > 0 \Longrightarrow \sum_{(i,j) \in T_m} x(i,j) = n_m, \; m = 1, 2, ..., z.$$

Алгоритъмът стартира от едно възможно решение на дуалната задача, в което дуалните променливи u_i и v_m удовлетворяват (8.19) и (8.21) (Началното сдвояване $M = \emptyset$). При всяка итерация на алгоритъма се изменят сдвояването M и (или) стойностите на дуалните променливи, така че ограниченията на двете задачи и условията (8.19), (8.21) да продължават да са изпълнени, а условието (8.20) да се изпълнява допълнително за поне още една двойнствена променлива u_i. Тъй като има n на брой двойнствени променливи u_i (колкото са върховете на графа), то след не повече от n итерации всички условия (8.20) също ще бъдат изпълнени. По този начин ще намерим допустими решения на правата и дуалната задача, за които са изпълнени всички условия (8.19) - (8.21). С други думи, ще получим сдвояване с максимално тегло.

Условието (8.20) — ако двойнствената променлива u_i за върха i има положителна стойност, то върхът i е наситен, се нарушава само за ненаситени върхове, на които съответстват положителни стойности на двойнствените променливи. Ето защо алгоритъмът по същество разглежда връх v, за който $u_v > 0$

и строи алтернативно дърво с корен в този връх v. Както вече видяхме в предишния алгоритъм, като резултат е възможна една от следните три ситуации:

а) намерена е аугментална верига;

б) намерен е цикъл с нечетна дължина;

в) построено е унгарско дърво.

1. В случая а) намерената аугментална верига е силна увеличаваща верига и това води до увеличение на общото тегло на сдвояването и до включване на ребро в сдвояването, инцидентно с v. По този начин за върха v се изпълнява условие (8.20).

2. В случая б) намереният нечетен цикъл се свива до фиктивен, псевдовръх и алгоритъмът продължава строенето на дървото.

3. В случая в) се променят стойностите на двойнствените променливи така, че ограниченията на правата и обратната задача да са налице, да са налице и ограниченията (8.19) и (8.21) с евентуално изключение, т.е. неизпълнение на условията (8.20). Като резултат в алтернативното дърво се добавя ребро, невключено в него и в крайна сметка или стойността на u_v се намалява до нула (изпълнява се условието (8.20)) или върхът v става наситен, т.е. инцидентен с ребро от сдвояването.

Псевдовърховете, получени в резултат на свиване на нечетни цикли, в последствие се "разгъват", като по този начин алгоритъмът поражда една последователност от графи G_0, G_1, ..., G_t. Ще дадем едно по-формално описание на този алгоритъм.

АЛГОРИТЪМ ЗА МАКСИМАЛНО ПО ТЕЖЕСТ СДВОЯВАНЕ. *СТЪПКА 1.* Нека началното сдвояване е $M_0 = \emptyset$, всички двойнствени променливи v_m са равни на нула, $m = 1, 2, ..., z$ и стойностите на двойнствените променливи u_i, $i \in X$ удовлетворяват $u_i + u_j \geq a(i,j)$ за всяко ребро (i,j) (На практика можете да изберете реброто с максимално тегло и всяко u_i да бъде половината от това тегло). Полагаме $k = 0$ и изходния граф обозначаваме с $G_k = (X_k, A_k)$.

СТЪПКА 2. В G_k се избира нефиктивен ненаситен връх v, за който $u_v > 0$. Ако такъв връх не съществува, GO TO *стъпка 6.* В противен случай в G_k се определя множеството A^* от ребра (i,j), за които

$$u_i + u_j + \sum_{(i,j)\in T_m} v_m = a(i,j). \tag{8.22}$$

В множеството A^* чрез алгоритъма за строене на алтернативно дърво се строи дърво с корен върхът v. Ако бъде на-

мерена аугментална верига GO TO *стъпка 3.* Ако се стигне до нечетен цикъл, GO TO *стъпка 4.* Ако бъде построено унгарско дърво, GO TO *стъпка 5.*

СТЪПКА 3. (Аугментация) Ребрата от аугменталната верига, участващи в M_k, се заменят с ребрата на веригата, неучастващи в M_k и обратно. Върхът v става наситен. GO TO *стъпка2.*

СТЪПКА 4. (Нечетен цикъл) Положете $k = k + 1$. Означете нечетния цикъл с C_k и го свийте до псевдовръх a_k. Означете новия граф с $G_k = (X_k, A_k)$. Означете с M_k сдвояването, включващо всички ребра, принадлежащи едновременно на G_k и M_{k-1}. Маркировките на всички върхове от цикъла са еднакви с маркировката на фиктивния връх a_k. Върнете се на *стъпка 2* и продължете строенето на алтернативно дърво с корен във връх v от G_k (v може да е фиктивен).

СТЪПКА 5. (Унгарско дърво) Означете с δ_1

$$\delta_1 = \min\{u_i + u_j - a(i,j)\}, \tag{8.23}$$

като минимумът се взима по всички (i,j), такива че $i \in X_0$ се явява out-връх за дървото и върхът $j \in X_0$ е немаркиран (неразгледан). Ако няма такива ребра (i,j), то $\delta_1 = \infty$.

Означете с δ_2

$$\delta_2 = \frac{1}{2}\min\{u_i + u_j - a(i,j)\}, \tag{8.24}$$

като минимумът се взима по всички (i,j), такива че $i \in X_0$ и $j \in X_0$ са out-върхове на дървото, които не са свити в един и същ фиктивен връх. Ако няма такива ребра (i,j), то $\delta_2 = \infty$.

Означете с δ_3

$$\delta_3 = \frac{1}{2}\min\{v_m\}, \tag{8.25}$$

където минимумът се взима по всички множества от върхове V_m, мощността на които е нечетно число и които са свити в псевдовръх a_k, който е in-връх на дървото. Ако няма такива върхове, то $\delta_3 = \infty$.

Означете с δ_4

$$\delta_4 = \min\{u_i\}, \tag{8.26}$$

където минимумът се взима по всички out-върхове на дървото $i \in X_0$. Ако няма такива върхове, то $\delta_4 = \infty$.

Означете с δ

$$\delta = \min\{\delta_1,\ \delta_2,\ \delta_3,\ \delta_4\}. \tag{8.27}$$

Актуализирайте стойностите на дуалните променливи u_i и v_m по следния начин:

а) променливите u_i, съответни на out-върхове на дървото, намалете с δ;

б) променливите u_i, съответни на in-върхове на дървото, увеличете с δ;

в) за всеки фиктивен out-връх в G_k увеличете неговата дуална променлива v_m с 2δ;

г) за всеки фиктивен in-връх в G_k намалете неговата дуална променлива v_m с 2δ.

Ако $\delta = \delta_1$, включете в A^* реброто (i, j), за което се достига минимумът в (8.23) (Това ребро сега може да бъде включено в строеното дърво, поради което се връщаме към *стъпка 2* и продължаваме строенето на алтернативно дърво с корен във върха v.

Ако $\delta = \delta_2$, в A^* се включва реброто (i, j), за което се достига минимумът в (8.24). Това ребро сега може да бъде включено в строеното алтернативно дърво, което ще води до получаване на нечетен цикъл, поради което се връщаме на *стъпка 2* и продължаваме строенето на алтернативно дърво с корен във върха v.

Ако $\delta = \delta_3$, някоя от дуалните променливи v_i става нула. "Разгънете" фиктивния връх, съответен на тази дуална променлива отново до изходния нечетен цикъл. Положете $k = k+1$. Означете получения граф с $G_k = (X_k, A_k)$. Означете с M_k сдвояването, състоящо се от ребрата на сдвояването M_{k-1} и онези n_i ребра от множеството T_i, които сдвояват $2n_i$ ненаситени върхове от V_i (оставащият връх в множеството V_i е сдвоен, инцидентен с ребро от M_k, тъй като всички фиктивни in-върхове в G_{k-1} са сдвоени, инцидентни с ребра от сдвояването M_{k-1}). Преминете към *стъпка 2* и продължете строенето на алтернативно дърво с корен във върха v.

Ако $\delta = \delta_4$, то дуалната променлива u_i, съответна на out-връх i, става равна на нула. Тогава в алтернативното дърво веригата, съединяваща корена v с върха i, се явява неутрална аугментална верига. "Направете" ребрата на веригата, неучастващи в M_k ребра на на сдвояването, а участващите в M_k — неучастващи. Очевидно върхът v става наситен връх в полученото сдвояване, а върхът i става ненаситен (неинцидентен с ребро от сдвояването), което е коректно поради $u_i = 0$.

СТЪПКА 6. (Разгъване на фиктивните върхове) Стъпката се изпълнява, само когато на *стъпка 2* са разгледани всички върхове, за които не е изпълнено условието (8.20). Разглеждат се всички останали фиктивни върхове в получения до този момент граф. В обратен ред на реда на получаването (по-късно полученият фиктивен връх се разгъва по-рано), фиктивните върхове се разгъват до съответните нечетни цикли и се образува максимално сдвояване за всеки от нечетните цикли. Така, след разгъването на всички фиктивни върхове, се получава сдвояване с максимално тегло за изходния граф G_0. Край.

От описанието на алгоритъма се вижда (вж. в *стъпка 5* случая $\delta = \delta_3$), че редът на разгъване на фиктивните върхове до нечетни цикли не съвпада с реда на тяхното формиране.

По-строга обосновка, доказателство за коректността на алгоритъма можете да намерите в [47] и др.

4. Оптимални покрития в графи

В параграф 1.5 на предишната глава показахме, че ако знаем алгоритъм за намиране на максимално по мощност сдвояване, лесно може да се намери покритие с минимална мощност. Беше предложена и конкретна процедура за това. Тъй като в този параграф изложихме алгоритми за намиране на максимално по мощност сдвояване, следва че с тях и цитираната по-горе процедура от параграф 1.5 можем да намираме минимални по мощност покрития в произволен граф. Ето защо ще разгледаме само въпроса за намиране на минимално по тежест покритие.

МИНИМАЛНО ПО ТЕЖЕСТ ПОКРИТИЕ. И в този случай, както при алгоритмите за максимални сдвоявания, основна процедура се явява процедурата за построяване на алтернативно дърво. Техниката, която се използва, се базира отново на връзката между решенията на правата и дуалната задача (вж. параграф 2.1 и предишния алгоритъм). Алгоритъмът е предложен от автора на [52].

Нека $G = (X, A)$ е произволен граф. Да означим, както преди, с $V = \{V_1, V_2, \ldots, V_z\}$ множеството от всички подмножества на множеството X, състоящи се от нечетен брой елементи. Да означим с $2n_m + 1$ броя на върховете във V_m, а с U_m — множеството от такива ребра, че поне един от инцидентните с тях върхове е от V_m. Ясно е, че *всяко покритие трябва да съдържа поне $n_m + 1$ ребра от U_m*.

Нека и сега означим с $a(i,j) > 0$ теглото на реброто (i,j), и нека $x(i,j) = 1$, ако реброто (i,j) е от покритието, и $x(i,j) = 0$, ако реброто не е от покритието.

Да формулираме и анализираме проблема за намиране покритие с минимално тегло като задача на линейното оптимиране, както това е направено в [47]:

$$\sum_{(i,j)} a(i,j)x(i,j) \longrightarrow \min \tag{8.28}$$

при ограничения

$$\sum_{j} [x(i,j) + x(j,i)] \geq 1, \quad \text{за } \forall i \in X, \tag{8.29}$$

$$\sum_{(i,j)\in U_m} x(i,j) \geq n_m + 1, \; m = 1, \; 2, \; ..., \; z, \tag{8.30}$$

$$x(i,j) \geq 0, \; \text{за всяко ребро } (i,j). \tag{8.31}$$

Очевидно всяко покритие удовлетворява ограниченията на тази задача. Нейната дуална задача е:

$$\sum_{i\in X} u_i + \sum_{m=1}^{z} (n_m + 1)v_m \longrightarrow \max \tag{8.32}$$

при ограничения

$$u_i + u_j + \sum_{m:(i,j)\in U_m} v_m \geq a(i,j), \; \text{за всички ребра } (i,j), \tag{8.33}$$

$$u_i \geq 0, \; \text{за } \forall i \in X, \tag{8.34}$$

$$v_m \geq 0, \; m = 1, \; 2, \; ..., \; z. \tag{8.35}$$

Условията за равновесие на правата и дуалната задача, които получихме в параграф 2.1, в случая са следните:

$$(8.36) \qquad x(i,j) > 0 \Longrightarrow u_i + u_j + \sum_{m:(i,j)\in U_m} v_m = a(i,j), \text{ за } \forall (i,j)$$

$$(8.37) \qquad u_i > 0 \Longrightarrow \sum_{j\in X}[x(i,j) + x(j,i)] = 1, \text{ за } \forall i \in X,$$

т.е. върхът i се покрива само от едно ребро.

$$(8.38) \qquad v_m > 0 \Longrightarrow \sum_{(i,j)\in U_m} x(i,j) = n_m + 1, \ m = 1,\ 2,\ ...,\ z.$$

Следователно във всяко множество върхове с нечетна мощност $2n_m + 1$ върха се покриват от n_m ребра, чиито краища са от множеството и от едно ребро, на което единият край принадлежи на множеството.

В дуалната задача променливата u_i е съответна на ограничението (8.29), т.е. u_i е дуална променлива за върха i. Дуалната променлива v_m, съответна на ограничението (8.30), е двойнствена за подмножеството върхове V_m, което е с нечетна мощност.

Аналогично на предишния алгоритъм и тук построените цикли с нечетна дължина се свиват във фиктивни върхове a_k. Всеки такъв псевдовръх a_k съдържа подмножество върхове с нечетна мощност. Дуалната променлива v_k, свързана с подмножеството, се разглежда като дуална за фиктивния връх a_k.

От ограниченията на дуалната задача следва, че

$$u_i \leq \min_j \{a(i,j),\ a(j,i)\}.$$

Когато за дуалната променлива u_i горното неравенство е изпълнено като равенство, върхът i се нарича *уплътнен*, а в противен случай — *неуплътнен*. Ако $u_i = 0$, върхът i се нарича *празен*. Ясно е, че ограничението (8.37) не касае празните върхове и само празните върхове могат да са инцидентни с повече от едно ребро от покритието. Освен това, всеки уплътнен връх трябва да бъде съединен с ребро с празен връх.

Алгоритъмът за построяване на покритие с минимално тегло се осъществява на два етапа. На първия етап се строи сдвояване, а на втория сдвояването се преобразува в покритие. При

стартирането на първия етап всички $x(i,j)$ са нула, дуалните променливи v_m са равни на нула, а стойностите на дуалните променливи u_i удовлетворяват ограниченията на дуалната задача. В първия етап, на всяка итерация се взима ненаситен out-връх v, който е неуплътнен. Като резултат от изпълнението на итерацията или ребро, инцидентно с v, се включва в сдвояването, или дуалните променливи се променят, така че да бъде уплътнен върха v. При това на първия етап ограниченията на дуалната задача трябва да остават удовлетворени, както и да са удовлетворени условията (8.36) и (8.38).

При втория етап най-напред се изключват ненаситените и неуплътнени върхове. След това полученото като резултат от първия етап сдвояване се преобразува в покритие, като в сдвояването се включват ребра, всяко от които съединява уплътнен връх със съответен празен връх. По този начин се получава покрите, за което са удовлетворени условията на правата и дуалната задача, както и условията (8.36) - (8.38), което гарантира минималност на покритието по отношение на теглото.

При изпълнението на първия етап на практика проблемът е как да се актуализира сдвояването и (или) дуалните променливи, за да може върхът да стане наситен или уплътнен. За целта е достатъчно в множеството ребра, удовлетворяващи ограничението (8.33) като равенство, да се построи алтернатвно дърво с корен във върха v.

1. Ако в резултат от прилагане на алгоритъма за строене на алтернативно дърво се получи аугментална верига, се прилага познатата процедура — светлите ребра на веригата стават тъмни (включват се в сдвояването), а тъмните — светли (изключват се от сдвояването). Върхът v става инцидентен с ребро от сдвояването.

2. Ако в резултат на строене на дървото се получи нечетен цикъл, той се стяга до фиктивен връх, след което продължава строенето на алтернативно дърво в получения граф.

3. Ако в резултат на строене на алтернативното дърво се стигне до унгарско дърво, дуалните променливи се актуализират, така че:

а) към построеното алтернативно дърво може да се добави ново ребро или

б) out-върхът j на дървото става уплътнен.

В случая а) продължава строенето на алтернативното дърво с корен върха v, а в случая б) е намерена неутрална аугментална верига от корена v до върха j. В аугменталната верига светлите ребра се "обръщат" в тъмни и обратно, в резултат на което коренът v става наситен за новото сдвояване, а върхът j

става ненаситен, уплътнен връх.

От казаното следва, че върхът v в крайна сметка става или наситен, или уплътнен.

След свиването на нечетните цикли във фиктивни върхове, алгоритъмът за построяване на покритие с минимално тегло преобразува обратно фиктивните върхове в съответните изходни нечетни цикли. Редът на разгъване на фиктивните върхове частично съответства на реда на тяхното получаване.

По-формално описание на този алгоритъм и неговата обосновка можете да намерите в [47] и [52].

Забележка: В случая, когато за теглата не ребрата са допустими отрицателни стойности, оптималното решение на задачата на линейното оптимиране (8.28) - (8.31) ще се достига при $x(i,j) = \infty$ при $a(i,j) < 0$. Очевидно всяко ребро с отрицателно тегло трябва задължително да участва в покритието с минимално тегло, а всяко ребро с нулево тегло може да бъде включено в покритето с минимално тегло. Ето защо, ако искаме коректно да използваме предложения алгоритъм, трябва да се направи следното. Достатъчно е ребрата с отрицателни тегла да се преобразуват като ребра с нулево тегло. След това с предложения алгоритъм може да се намери покритие с минимално тегло в получения граф с неотрицателни тегла на ребрата. Към полученото покритие се добавят всички неучастващи в покритието ребра, имащи до преобразуването отрицателни тегла. В резултат се получава покритие с минимално тегло за изходния граф.

2.9. Алгоритми за задачи от тип CPP

В този параграф ще продължим изследванията, започнати в параграф 1.6 на предишната глава. Там дефинирахме понятието ойлеров цикъл (верига) в неориентиран граф $G = (X, A)$ и дадохме алгоритъм за построяване на покриващ графа ойлеров цикъл — цикъл, който минава по всяко ребро на графа точно веднъж. Припомняме, че необходимо и достатъчно условие за съществуване на такъв цикъл е всички върхове на графа да бъдат от четна степен.

Както вече изяснихме в параграф 1.6, много задачи от практиката се свеждат до търсене на оптимални маршрути в графи, при това в повечето случаи не е налице условието всички върхове на графа да са от четна степен.

В този параграф ще дадем алгоритми за намиране на решение

на така наречената *задача за китайския пощальон.* За краткост ще означаваме тази задача и задачите от този клас с абревиатурата CPP (англ. chinese postman problem). Класическата формулировка на тази задача е: *"Пощальон трябва да тръгне от пощенския офис и минавайки по всички адреси, да се върне отново в офиса, така че да измине минимално разстояние".* На езика на графите формулировката на проблема изглежда така:

ЗАДАЧА ОТ ТИП CPP. В граф G да се намери затворен маршрут с минимална дължина, включващ всички ребра на графа, т.е. по всяко ребро на графа да се премине поне веднъж, като движението завърши в изходната точка s.

Очевидно, когато всички върхове на графа са четни (четен граф), съществува ойлеров цикъл, покриващ графа G, т.е. цикъл, минаващ по всяко ребро точно веднъж и дължината на ойлеровия цикъл се явява решение на CPP.

Когато съществуват върхове с нечетна степен, в графа няма ойлеров цикъл. Това означава, че някои от ребрата на графа трябва повторно да бъдат обхождани, за да се върнем в изходния връх s. В този случай проблемът се състои в това кои от ребрата повторно да се обходят, така че в крайна сметка изминатото разстояние да бъде минимално. С други думи CPP е една типична оптимизационна задача.

При по-нататъшното изложение, ако не е казано друго, ще считаме, че графът G е свързан и теглата $a(i,j)$ на ребрата са положителни (макар че част от получените резултати ще се отнасят и за свързани графи с отрицателни тегла).

Лесно се съобразява, че ако бъде намерен оптимален маршрут на пощальона с начало и край връх s, дължината на оптималния маршрут в графа не зависи от избора на начален връх. Това е така, защото ако е избран за начало на маршрута върха s, ние рано или късно ще преминем през друг връх t, различен от s, откъдето отново ще се върнем в s, т.е. оптималният маршрут с начало връх t има същата дължина като оптималния маршрут с начало s.

1. CPP в неориентиран граф $G = (X, A)$.

1. Когато графът G е четен, както вече споменахме, в него има ойлеров цикъл, който дава решение на CPP. Алгоритъм за намиране на ойлеров цикъл в неориентиран граф с четни степени на върховете, беше даден в параграф 1.6.

2. Нека графът $G = (X, A)$ не е четен. При всеки маршрут на

пощальона за всеки връх x броят на "влизанията" във върха е равен на броя на "излизанията" от него. Оттук веднага следва, че ако връх x е нечетен (с нечетна степен), то поне едно ребро, инцидентно с този връх ще бъде обходено повторно. Ако с $l(i,j)$ означим броя на допълнителните преминавания по реброто (i,j), пощальонът ще преминава през реброто (i,j), $l(i,j)+1$ пъти. Повторното (многократното) преминаване през реброто (i,j) може да се интерпретира като наличие на ново, фиктивно ребро (i,j) в графа G (или нови фиктивни ребра (i,j)). С други думи, пощальонът трябва да определи стойностите на променливите $l(i,j)$ така, че $\sum a(i,j).l(i,j)$ да бъде минимално. Припомняме, че броят на нечетните върхове във всеки граф е четно число (теорема 1.1 от глава I).

Нека върхът x е нечетен и повторно се обхожда реброто (x,y). Добавянето на това фиктивно ребро (x,y) в графа G прави четна степента на x и променя степента на връх y — ако върхът y е бил четен, след добавянето на фиктивното ребро (x,y) степента му ще стане нечетна. За да изравним броя на "влизанията" и "излизанията" във върха y, трябва инцидентно с y ребро да бъде повторно обходено и т.н., докато стигнем до друг връх z, който е от нечетна степен (добавянето на фиктивно ребро, инцидентно със z, го прави четен връх).

С други думи, за да се изравни броят на "влизанията" и "излизанията", за всеки връх на графа:

а) ако върхът x е нечетен — нечетен брой ребра, инцидентни с този връх, трябва повторно да се обходят;

б) ако върхът x е четен — четен брой ребра, инцидентни с този връх (нулата също е четно число) трябва повторно да се обходят.

От казаното следва, че всяка верига от повторни, фиктивни ребра, стартираща от нечетния връх x, задължително завършва в друг връх с нечетна степен, като разбира се веригата може да преминава през върхове с четна степен.

Да означим с X_{odd} множеството от всички нечетни върхове в графа G. Нека P е множеството от вериги ρ_{ij} с крайни върхове x_i и $x_j \in X_{odd}$, такива че никои две вериги нямат общ краен връх и тези вериги покриват върховете X_{odd}. Броят на тези вериги очевидно е $\frac{1}{2}|X_{odd}|$. По друг начин казано, P е съвършено сдвояване на върховете X_{odd} чрез вериги. Ако добавим ребрата на веригата ρ_{ij} като фиктивни, паралелни ребра в графа G, очевидно ще "удвоим" някои от ребрата на G. Изпълнявайки това за всяка верига $\rho_{ij} \in P$, ще получим s-граф $G(P)$. Тъй като някои

ребра на G могат да участват в различни вериги ρ_{ij}, то някои от ребрата могат да се окажат утроени, учетворени и т.н. От направените коментари следва очевидно верността на следното твърдение:

▷ *ТЕОРЕМА 9.1. За всеки цикъл, покриващ G, може да се избере множество P, за което в графа $G(P)$ има ойлеров покриващ цикъл, съответен на първоначалния избран цикъл в G. При това съответствието е такова, че ако първоначалният цикъл минава по реброто (x_i, x_j) в G, l пъти, то в $G(P)$ съществуват l ребра (реалното и $l-1$ фиктивни) между x_i и x_j, всяко от които се обхожда точно един път в ойлеровия цикъл в $G(P)$. Вярно е и обратното твърдение.* ◁

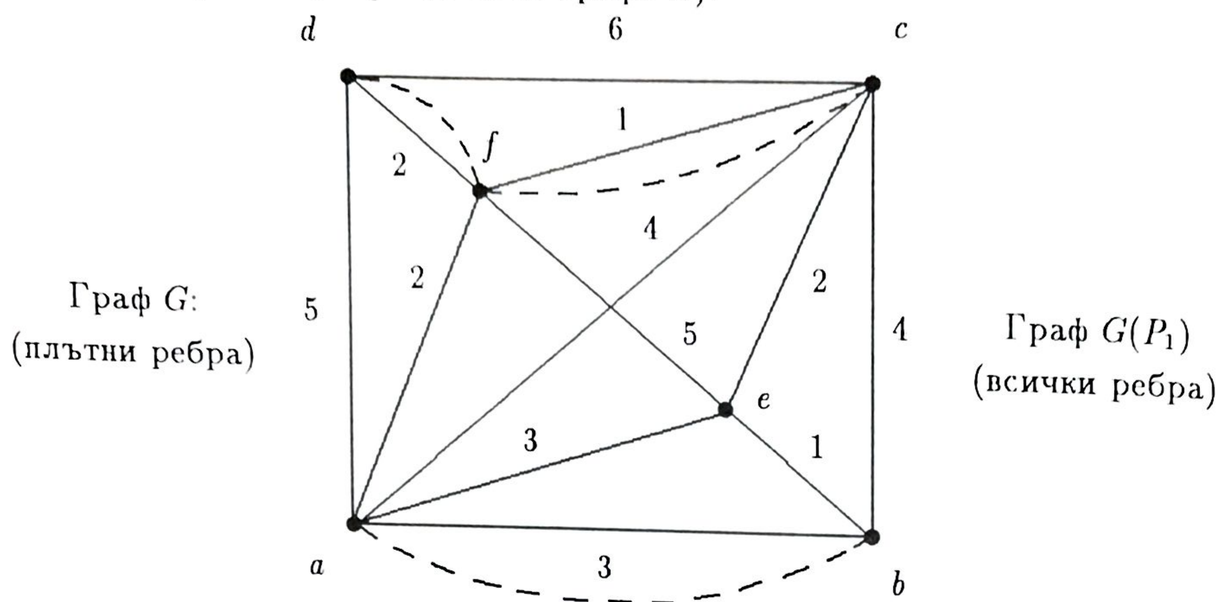
Доказаната теорема автоматично дава идея за решаване на СРР. От теоремата следва, че за намиране на оптимален маршрут, т.е. решение на СРР, е достатъчно да намерим онова "верижно" сдвояване P^* на върховете от X_{odd}, което е с минимално сумарно тегло, т.е. даващо минимално допълнително тегло. Намирането на това верижно сдвояване може да стане така:

АЛГОРИТЪМ ЗА РЕШЕНИЕ НА СРР. Разглеждаме графа $G' = (X_{odd}, A')$, където X_{odd} е множеството върхове с нечетна степен от G, а A' е множество ребра, съединяващи всяка двойка върхове от X_{odd}, т.е. пълен граф с върхове X_{odd}.

С помощта на алгоритъма на Флойд (параграф 2.3) намираме най-кратките пътища между всеки два върха на графа G' и техните дължини. Да означим с d_{ij} най-краткия път между нечетните върхове x_i и x_j, т.е. теглото на реброто (x_i, x_j) в G'. Тъй като в G' трябва да намерим сдвояване с минимално сумарно тегло, заменяме теглото d_{ij} на всяко ребро в графа G' с $\theta - d_{ij}$, където θ е достатъчно голямо число и прилагаме алгоритъма за търсене на сдвояване с максимално тегло, предложен в параграф 2.8. Очевидно този алгоритъм ще генерира сдвояване с минимално сумарно тегло за графа G' при тегла на ребрата d_{ij}.

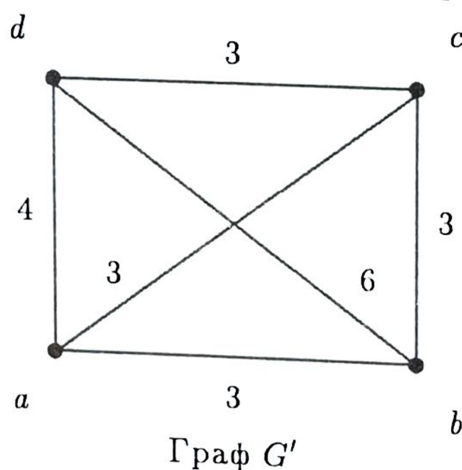
Тъй като ребрата в G' са съответни на веригите, съединяващи нечетните върхове на G, определянето на верижно сдвояване P^* с минимално тегло задава ребрата в графа G, които допълнително ще бъдат обходени, така че полученият маршрут да бъде оптимален.

ПРИМЕР 9.1. Да се намери оптималния маршрут на пощальона в следния граф (пунктираните ребра не са от графа G):



Най-кратките разстояния между всеки два нечетни върха в графа G са зададени в таблицата долу:

	a	b	c	d
a	0	3	3	4
b	3	0	3	6
c	3	3	0	3
d	4	6	3	0



Граф G'

Теглата на ребрата в G' са дължините на най-кратките пътища между всеки два нечетни върха на графа G. В конкретния пример няма да актуализираме тези тегла d_{ij}, заменяйки ги с $\theta - d_{ij}$ (напр. $\theta = 100$) и да прилагаме алгоритъма за сдвояване с максимално тегло (който ще генерира в G' с тегла d_{ij} сдвояване с минимално тегло), тъй като случаят е достатъчно прост. Всевъзможните сдвоявания в G' (без да актуализираме теглата) са:

$P_1 = \{(a,b),\ (d,c)\}$ — тегло на сдвояването $3 + 3 = 6$.

Веригата с минимално тегло, сдвояваща върховете a и b е реброто (a,b), а веригата с минимално тегло, сдвояваща върховете d и c е (d,f), (f,c).

$P_2 = \{(a,d),\ (b,c)\}$ — тегло на сдвояването $4 + 3 = 7$.

$P_3 = \{(a,c),\ (b,d)\}$ — тегло на сдвояването $3 + 6 = 9$.

Сдвояването с минимално тегло е P_1. Следователно добавянето на ребрата от това верижно сдвояване към графа G ще доведе до графа $G(P_1)$, в който всички върхове са от четна степен. Ойлеровият цикъл в графа $G(P_1)$ задава оптималния маршрут на пощальона.

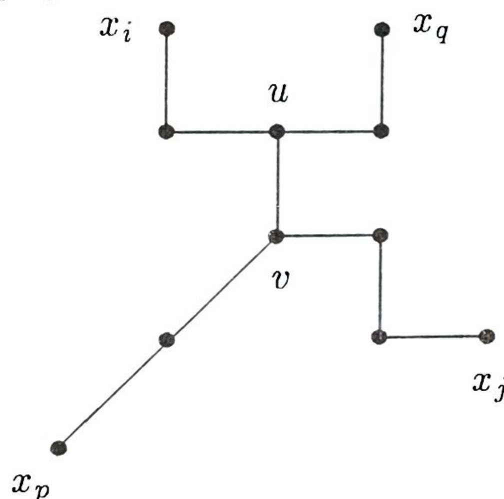
С предложения в параграф 1.6 алгоритъм за намиране на ойлеров цикъл, намерете този цикъл в получения граф $G(P_1)$.

Ще докажем следното твърдение:

▷ *ТЕОРЕМА 9.2. Нека P^* е верижното сдвояване на нечетните върхове на G с минимално тегло. Тогава в графа $G(P^*)$ за всяко реално ребро на графа G съществува най-много едно фиктивно ребро.*

Доказателство: Да разгледаме в P^* две произволни вериги ρ_{ij} и ρ_{pq}, свързващи съответно върховете x_i, x_j и x_p, x_q. Напомняме, че това са най-кратките $(x_i - x_j)$ и $(x_p - x_q)$ пътища и освен това, че P^* е верижното сдвояване с минимално тегло.

Ще покажем, че никои две вериги на P^* не могат да имат общо ребро, откъдето следва, че никое ребро на графа G не може да бъде обхождано допълнително повече от един път. Да допуснем, че веригите ρ_{ij} и ρ_{pq} от P^* са ребрено пресичащи се, т.е. имат общо ребро, както е показано на чертежа долу:



$$\rho_{ij} : \; x_i, \; \ldots, \; u, \; v, \; \ldots, \; x_j,$$

$$\rho_{pq} : \; x_p, \; \ldots, \; v, \; u, \; \ldots, \; x_q.$$

Тогава очевидно сумарното тегло на това верижно сдвояване ще включва теглата на всички ребра по веднъж плюс теглото $a(u, v)$ на реброто (u, v). Очевидно обаче, в следствие допускането, съществува друго верижно сдвояване на тези двойки нечетни върхове, например

$$\rho_{iq} : \; x_i, \; \ldots, \; u, \; \ldots, \; x_q,$$

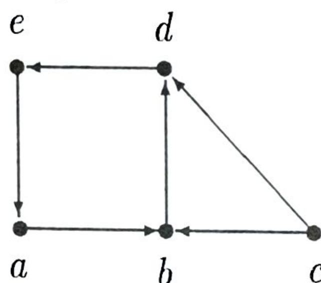
$$\rho_{pj} : \; x_p, \; \ldots, \; v, \; \ldots, \; x_j.$$

Теглото на това сдвояване е по-малко от теглото на предишното и то точно с $a(u, v)$, което противоречи на това, че P^* е верижно сдвояване с минимално сумарно тегло. ◁

От теоремата следва, че след прилагането на алгоритъма за намиране на оптимален маршрут на пощальона, всяко ребро на графа G се обхожда поне веднъж и най-много два пъти.

2. CPP в ориентиран граф $G = (X, E)$.

При ориентирани графи задачата CPP (за разлика от неориентирани графи) може да няма решение. Това се случва за ориентирани графи, в които има такова множество от върхове $X' \subset X$, че липсват дъги, изходящи от върховете на множеството X' към върхове, непринадлежащи на X'. Например в графа, изобразен долу, CPP няма решение:



Множеството $X' = \{a, b, e, d\}$ е такова, че отсъстват дъги с начало връх от това множество и край в множеството върхове $X - X'$.

Ако не съществуват такива множества от върхове X' в графа G, задачата за пощальона винаги има решение.

В ориентирани графи, както и при неориентираните, броят "влизания" на пощальона в даден връх трябва да бъде равен на броя "излизания" от този връх. С други думи, ако полустепента на входа $d^-(x)$ на произволен връх x не съвпада с полустепента на изхода $d^+(x)$ на този връх, пощальонът задължително ще обхожда някои от дъгите, инцидентни с x повторно. По-точно:

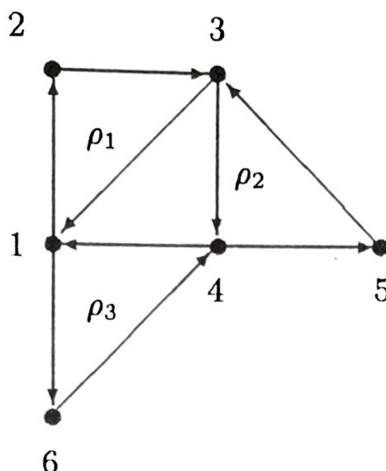
а) ако $d^+(x) > d^-(x)$, пощальонът е длъжен да обходи повече от един път някои от влизащите във върха x дъги;

б) ако $d^-(x) > d^+(x)$, пощальонът е длъжен да обходи някои от излизащите от x дъги повече от един път.

ПЪРВИ СЛУЧАЙ. Графът $G = (X, E)$ е *симетричен*, т.е. $d^+(x) = d^-(x)$ за всеки връх $x \in X$.

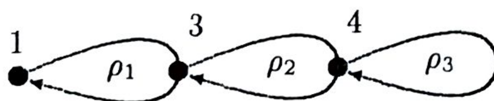
Очевидно в този случай графът G е четен, т.е. степента на всеки негов връх е четна и за намиране на оптимален маршрут може да се използва методът за намиране на ойлеров цикъл, предложен в параграф 1.6. С други думи, при симетрични ориентирани графи оптималното решение на CPP се явява съществуващия в графа покриващ ойлеров маршрут.

ПРИМЕР 9.2. Да се намери оптимален маршрут на пощальона в следния граф:



Използваме предложения метод в параграф 1.6. Тръгваме от върха 1 и се движим (в случая съобразявайки се с посоката на ребрата) по различни дъги, докато отново стигнем във върха 1 (всеки друг връх на графа можеше да бъде избран за начален), т.е. докато получим някакъв ойлеров цикъл (незадължително покриващ). Например цикълът $\rho_1 = (1,2),\ (2,3),\ (3,1)$.

Ако този ойлеров цикъл е покриващ за графа, задачата е решена. В противен случай, дъгите на получения цикъл се изключват (от разглеждане) от графа — както е в нашия случай, цикълът ρ_1 не е покриващ ойлеров цикъл. Избираме друг произволен връх, инцидентен с някоя от останалите дъги в графа. Например връх 3 и по същия начин се движим, докато получим нов цикъл $\rho_2 = (3,4),\ (4,5),\ (5,3)$. Изключваме и тези дъги от графа (правим това, докато множеството на оставащите дъги е $\neq \emptyset$). След това избираме връх 4 (инцидентен с оставащи, неотстранени дъги) и получаваме цикъла $\rho_3 = (4,1),\ (1,6),\ (6,4)$. Отстраняваме дъгите на цикъла ρ_3 и в този момент всички дъги на графа вече са отстранени. По този начин разбиваме графа на прости ребрено непресичащи се цикли, което дава възможност бързо да получим покриващия ойлеров цикъл (за подробности виж параграф 1.6).



Движим се в първия цикъл ρ_1, докато стигнем началото на следващия цикъл ρ_2. Напускаме цикъла ρ_1 и продължаваме движението в цикъла ρ_2, докато стигнем до началото на следващия цикъл. Всеки път напускаме текущия цикъл щом стигнем до началото на следващия и т.н. докато стигнем до последния цикъл, който описваме изцяло, след това се връщаме в предпоследния цикъл и се движим по непреминатите негови дъги и т.н., докато достигнем до първия цикъл и завършим движението си в началния връх.

В конкретния случай полученият с помощта на циклите ρ_1, ρ_2 и ρ_3 ойлеров цикъл е следният:

$$(1,2),\ (2,3),\ (3,4),\ (4,1),\ (1,6),\ (6,4),\ (4,5),\ (5,3),\ (3,1).$$

ВТОРИ СЛУЧАЙ. Нека сега графът G не е симетричен, т.е. съществуват върхове $x \in X$, такива че $d^+(x) \neq d^-(x)$.

Както изяснихме в предварителните коментари, в този случай някои от дъгите на графа G ще бъдат обхождани повторно (неколкократно). С помощта на потоковите алгоритми може да се реши задачата CPP за оптимален маршрут на пощальона по следния начин:

ИДЕЯ НА АЛГОРИТЪМА. Да означим с $f(i,j)$ броя на повторните обхождания на дъгата (i,j). Очевидно $f(i,j) \geq 0$, цяло число. Задачата за оптимален маршрут може да се формулира така:

$$(9.1) \qquad \sum a(i,j).f(i,j) \longrightarrow \min$$

при ограничения

$$(9.2) \qquad d^+(i) + \sum_j f(i,j) = d^-(i) + \sum_j f(j,i).$$

Ограниченията (9.2) изразяват условието във всеки връх да се влиза толкова пъти, колкото се излиза, като при това се минимизира изразът в (9.1). Условията (9.2) могат да се запишат още така:

$$(9.3) \qquad \sum_j [f(i,j) - f(j,i)] = d^-(i) - d^+(i) = D(i).$$

Формулираната задача (9.1), (9.3) представлява задача за поток с минимална стойност (вж. параграф 2.5).

Върховете, за които $D(i) > 0$, представляват източници с предлагане $D(i)$. Аналогично, върховете, за които $D(i) < 0$ се явяват стокове (крайни пунктове) с търсене, равно на $|D(i)|$. Върховете, за които $D(i) = 0$, се явяват вътрешни върхове на мрежата.

Формулираната задача за поток с минимална стойност в мрежа с няколко източника и няколко стока, както отбелязахме в параграф 2.5, може да се реши като се въведе допълнително главен източник S и главен сток T в мрежата. Допълнителният източник S е свързан с дъги с всички други източници на мрежата, а всички стокове (крайни пунктове) са свързани с дъги с главния сток T. Капацитетите (пропускателните способности) на дъгите, излизащи от главния източник S са равни на

предлагането на съответния (неглавен) източник s_i. Аналогично капацитетите на дъгите, влизащи в главния сток T, са равни на търсенето на съответния (не главен) сток t_j.

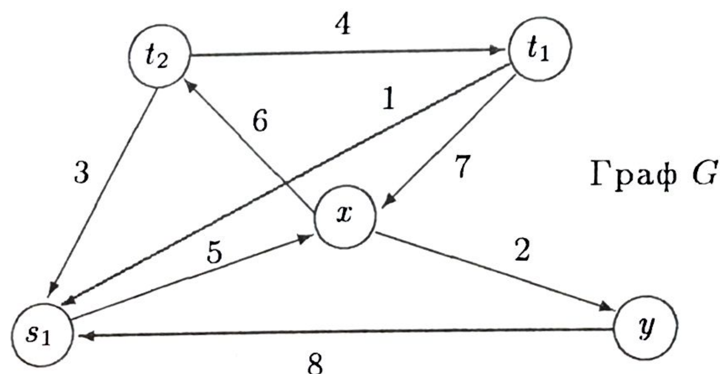
В получената по гореописания начин мрежа теглата (цените) на допълнителните дъги от вида (S, s_i) и (t_j, T) са нула, теглата на останалите дъги, т.е. дъгите на графа G, се разглеждат като цени за пренос на потока (капацитетите на тези дъги са ∞, т.е. са неограничени).

Тъй като десните части на равенствата (9.3) са цели числа, то алгоритъмът за поток с минимална стойност ще генерира в мрежата целочислен, неотрицателен поток $f(i,j)$ във всяка дъга (i,j).

Ясно е, че оптималните значения $f(i,j)$ на потока минимизират израза (9.1), ето защо след определнето на оптималните $f(i,j)$ можем да построим граф G', в който реалната дъга (i,j) на графа G повторно се обхожда още $f(i,j)$ пъти. Поради условието (9.3), еквивалентно на условието (9.2), графът G' се явява симетричен, т.е. в него броят на влизащите и излизащите дъги от всеки връх е равен. От разглежданията, направени в предишния случай, в G' може да се намери покриващ ойлеров маршрут и той ще се явява оптимално решение на задачата CPP за графа G.

Ще илюстрираме предложения алгоритъм със следния пример:

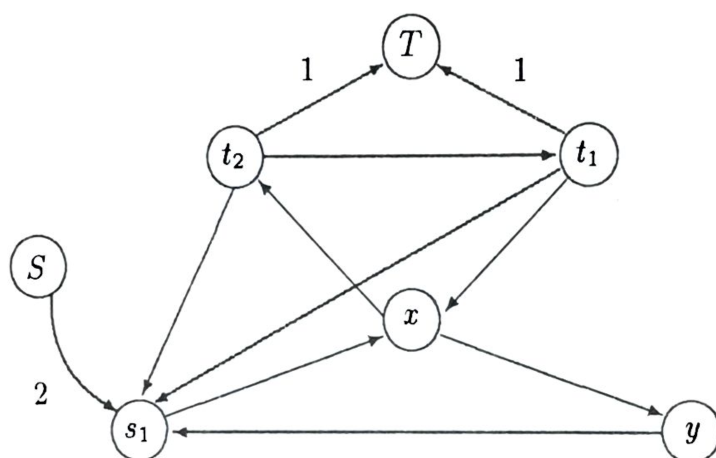
ПРИМЕР 9.3. Ще намерим оптимален маршрут на пощальона за следния ориентиран, несиметричен граф G. Теглата на дъгите в графа G са дадени на чертежа долу.



Пресмятаме полустепените на входа и изхода за всеки от върховете на графа G, за да определим съответно източниците, стоковете, вътрешните върхове на мрежата и капацитетите на допълнителните дъги, инцидентни с главния източник S и главния сток T, които в последствие ще въведем.

$d^-(s_1) = 3 > d^+(s_1) = 1$ — върхът s_1 е източник с предлагане $3 - 1 = 2$;
$d^-(x) = 2 = d^+(x)$ — върхът x е вътрешен връх за мрежата;
$d^-(y) = 1 = d^+(y)$ — върхът y е вътрешен връх за мрежата;
$d^-(t_1) = 1 < d^+(t_1) = 2$ — върхът t_1 е сток с търсене $|1 - 2| = 1$;
$d^-(t_2) = 1 < d^+(t_2) = 2$ — върхът t_2 е сток с търсене $|1 - 2| = 1$.

Въвеждаме допълнителен главен източник S и главен сток T. На чертежа долу е изобразена получената мрежа:



Числата, приписани на допълнителните дъги (S, s_1), (t_1, T) и (t_2, T) са намерените за тези дъги капацитети. Тези дъги са с тегло (цена) нула. Всички останали дъги в мрежата са с неограничен капацитет и тегла (цени), указани в предишния чертеж.

Алгоритъмът за търсене на поток с минимална стойност от S до T, очевидно ще пренесе една единица поток по веригата

$$(S, s_1),\ (s_1, x),\ (x, t_2),\ (t_2, T)$$

на сумарна цена 11.

Втората единица поток ще бъде пренесена по веригата

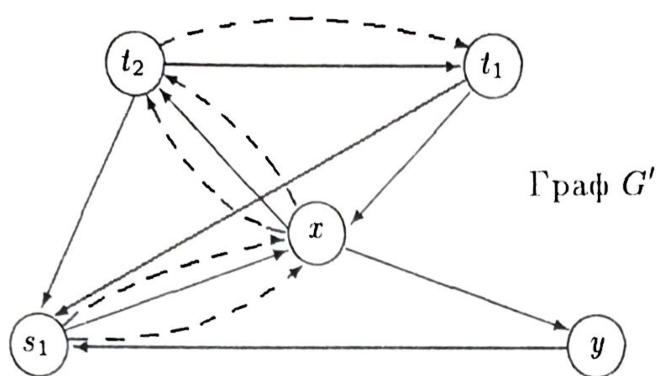
$$(S, s_1),\ (s_1, x),\ (x, t_2),\ (t_2, t_1),\ (t_1, T)$$

на сумарна цена 15.

Следователно $f(s_1, x) = 2$, $f(x, t_2) = 2$, $f(t_2, t_1) = 1$. Във всички други дъги на графа G величината на потока $f(i, j) = 0$.

Следователно пощальонът трябва допълнително да обходи дъгата (s_1, x) два пъти, дъгата (x, t_2) — два пъти и дъгата (t_2, t_1) — един път.

Добавянето на цитираните по-горе фиктивни дъги толкова пъти, колкото уточнихме, към графа G води до симетричен ориентиран граф G', за който вече дадохме алгоритъм, намиращ оптимален маршрут на пощальона.



Решаването на CPP в смесен граф, т.е. граф с ребра и дъги, се решава също с помощта на потокови алгоритми. Ще отбележим, че при смесен граф G трябва да се разгледат следните три случая:

1. Графът G е четен и симетричен.
2. Графът G е четен, но несиметричен.
3. Графът G е нечетен и несиметричен.

Първият от горните три случая лесно се решава като се приложат последователно дадените вече методи за симетричен ориентиран и четен неориентиран граф.

Алгоритми, решаващи проблема в другите два случая, можете да намерите в [47] и др.

2.10. Алгоритми за задачи от тип TSP

В параграф 1.6. на предишната глава формулирахме понятията хамилтонова верига и хамилтонов цикъл в граф G — цикъл, който минава през всички върхове на графа точно веднъж. В практиката съществува един голям клас задачи, свързани с намиране на оптимален маршрут в граф. Поради различната терминология, използвана в литературните източници, ще формулираме следната задача, която ще наречем *(обща) задача на търговския пътник (англ.* Travelling Salesman Problem*)*. За краткост тази задача ще означаваме с абревиатурата TSP.

1. Задача от тип TSP.

Търговец трябва да постои всеки град от даден регион (поне веднъж) и да се върне там, откъдето е тръгнал, като при това измине минимално разстояние (*).

Обърнете внимание, че при TSP няма изискване всеки град да е посетен точно веднъж. Търговецът се стреми да осигу-

ри поне едно преминаване през всеки град, като основният му проблем се състои в минимизиране на дължината на пътя, т.е. за него не е съществено дали ще преминава повече от един път през даден град. С други думи, ако минимизирането на изминатия път предполага многократни преминавания през даден град, търговецът ще ги реализира.

Обръщаме внимание, че в TSP дължината на пътя може да се интерпретира като разстояние в километри, цена на загуби в лева, загуба на време в часове и т.н.

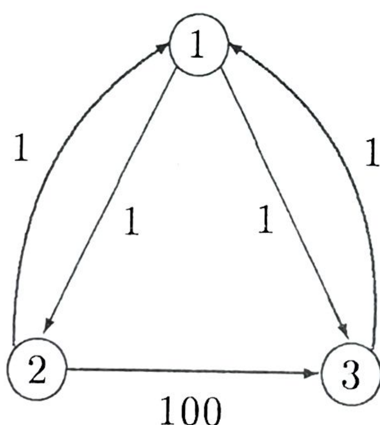
Ще формулираме и следната задача:

(**) В граф да се намери хамилтонов цикъл с минимална сумарна дължина.

В по-нататъшните разглеждания, ако не е казано нищо друго, ще разглеждаме свързани графи с тегла на дъгите $a(x, y) \geq 0$.

Очевидна е връзката и различията на формулираните задачи (*) и (**).

1. Обръщаме внимание, че оптималният маршрут на търговеца не винаги се явява хамилтонов цикъл.



В горния граф оптималният маршрут на търговеца очевидно е

$$(1, 2),\ (2, 1),\ (1, 3),\ (3, 1).$$

Търговецът е посетил всички градове и е изминал възможното минимално разстояние 4. Очевидно оптималният маршрут на търговеца не е хамилтонов цикъл. Оптималният хамилтонов цикъл (в случая той е един) е

$$(1, 2),\ (2, 3),\ (3, 1)$$

и неговата дължина е 102.

2. Има обаче връзка между формулираните по-горе две задачи и тя се дава със следната теорема:

▷ *ТЕОРЕМА 10.1. Ако за всяка двойка върхове (x, y) в графа G е изпълнено неравенството на триъгълника, т.е.*

$$a(x,y) \leq a(x,z) + a(z,y), \text{ за } \forall z \neq x, \ y,$$

то хамилтоновият контур се явява решение на TSP *(когато такова съществува).* ◁

С други думи, ако графът G удовлетворява неравенството на триъгълника, оптималното решение на (**) се явява решение на (*).

Не е необходимо да се търсят алгоритми за решаване на всяка от горните две задачи. Ако графът G не удовлетворява неравенството на триъгълника, дължината $a(x, y)$ на всяка дъга (x, y), неудовлетворяваща неравенството, можете да замените с дължината на най-краткия $(x - y)$ път. Тогава алгоритъмът, намиращ оптимален хамилтонов цикъл (т.е. алгоритъмът за решаването на (**)) ще генерира решение на (*) — достатъчно е в това решение дъгите $a(x, y)$ с намалена дължина да замените с дъгите, участващи в най-краткия път между върховете x и y. Ето защо е достатъчно да се намери алгоритъм за решаването на (**), което автоматично дава възможност да се решава и (*).

Обърнете внимание, че ако трябва да се намери хамилтонов цикъл с максимална дължина, също не е необходимо да се разработва алгоритъм за решаването на този проблем. В този случай може да се постъпи така (тази техника вече използвахме в параграф 2.8 и 2.9): Заменете теглата $a(x, y)$ на дъгите по следния начин

$$a'(x,y) = \theta - a(x,y) \geq 0, \text{ за } \forall (x,y),$$

където θ е достатъчно голямо число. Очевидно тогава

$$\sum_{(x,y)\in C} a'(x,y) = \sum_{(x,y)\in C} [\theta - a(x,y)] = \theta|X| - \sum_{(x,y)\in C} a(x,y),$$

където със C сме означили оптималния хамилтонов цикъл, а $|X|$ е броят на дъгите в цикъла (всеки хамилтонов цикъл в графа $G = (X, E)$ съдържа точно $|X|$ дъги). Ясно е, че при предложената процедура хамилтоновият цикъл с минимална дължина, състоящ се от дъгите с променени тегла е еквивалентен на хамилтоновия цикъл с максимална дължина, състоящ се от същите дъги с непроменена дължина.

В параграф 1.6 и 1.8 дадохме вече условия за съществуване на хамилтонови вериги (цикли) и алгоритми за тяхното намиране. Няма да повтаряме казаното, но в тази връзка ще добавим следното.

Ако графът не е силно свързан (вж. параграф 1.3), в него липсват хамилтонови цикли (хамилтоновият цикъл съдържа път между всяка двойка върхове на графа).

▷ *ТЕОРЕМА 10.2. Ако графът $G = (X, E)$ удовлетворява условията:*

1. Графът G е силно свързан;

2. $d(x) \geq |X|$, за $\forall x \in X$,

то в графа G съществува хамилтонов цикъл. ◁

Формулираната теорема е удобен начин за установяване съществуването на хамилтонов цикъл, тъй като с алгоритъма на Флойд или Данциг за НКП (вж. параграф 2.3) лесно се проверява условие 1., а условие 2. се проверява тривиално.

В бъдеще ще разглеждаме графи без примки и паралелни дъги. Хамилтоновите цикли не могат да съдържат примки, ето защо добавянето или премахването на примки в графа не влияе на съществуването на хамилтонов цикъл. Аналогично съществуването на паралелни дъги (x, y) не влияе на съществуването на хамилтонов цикъл и отстраняването на всички такива дъги с изключение на минималната дъга (x, y) не влияе на дължината на оптималния хамилтонов цикъл.

2. Методи за решаване на TSP. Долни граници

В параграф 1.8 споменахме за използването на долни граници като удобно средство, за да бъде отчетено отклонението на произволно допустимо решение от оптималното.

В [1] с използването на потокови алгоритми е даден метод за изчисление на долна граница L_1 за TSP.

Ние ще разгледаме начин за определяне долна граница за TSP с помощта на задачата за назначенията (англ. Assignment Problem) и задачата за покриващо дърво с минимално тегло (англ. Shortest Spanning Tree).

За краткост в бъдеще за цитираните горе две задачи ще използваме абревиатурите AP — задача за назначенията и SST — задача за покриващо дърво с минимално тегло.

Методите, които ще разгледаме, са предложени от Кристофидис Н. в [2].

Задачите AP и SST, които вече разгледахме в параграфи 1.5, 2.2 и 2.8, се решават просто и ефективно за разлика от задачата TSP. Връзката на тези две задачи с TSP дава възможност да се разработят ефективни методи за решаването на TSP.

ДОЛНА ГРАНИЦА НА TSP ЧРЕЗ AP. Задачата AP като задача на линейното оптимиране може да се формулира така

$$z = \sum_{j=1}^{n}\sum_{i=1}^{r} c_{ij}.\xi_{ij} \longrightarrow \min \quad (10.1)$$

при условия

$$\sum_{i} \xi_{ij} = \sum_{j} \xi_{ij} = 1, \text{ за } \forall i,\ j = 1,\ 2,\ \ldots,\ n, \quad (10.2)$$

$$\xi_{ij} = 0 \text{ или } 1. \quad (10.3)$$

Условието (10.2) гарантира цикличност на решението — във всеки връх влиза и от него излиза една дъга. Освен това $C = [c_{ij}]$ е матрицата на теглата, а $[\xi_{ij}]$ е $(n \times n)$-матрица, в която $\xi_{ij} = 1$, ако върхът x_i е "назначен към върха x_j (работникът x_i върши работата x_j) и $\xi_{ij} = 0$, в противен случай.

Очевидно същата схема може да се използва и при TSP като приемем $\xi_{ij} = 1$, ако търговецът преминава директно от град x_i в град x_j и $\xi_{ij} = 0$ в противен случай. Разбира се, в TSP трябва да се положи $c_{ii} = \infty$, за $i = 1,\ 2,\ \ldots,\ n$, за да отстраним примките.

От казаното е ясно, че ако към задачата AP, формулирана с (10.1) - (10.3), наложим допълнителното ограничение решението на AP да бъде единствен цикъл (хамилтонов), а не няколко несвързани цикъла, ще получим формулировка на задачата TSP.

С други думи, TSP се получава от AP с добавяне на допълнително ограничение (изискване). Добавянето обаче на произволно ограничение в AP увеличава или в най-добрия случай запазва минималното значение на z, определено в AP. Следователно $z_{\min}$ на AP се явява долна граница за решението на задачата TSP с матрица на теглата $[c_{ij}]$, т.е.

$$Optimum(AP) \leq Optimum(TSP).$$

ДОЛНА ГРАНИЦА НА TSP ЧРЕЗ SST. Нека $G = (X, A)$ е неориентиран граф (матрицата на теглата C е симетрична). Ако реброто (x_1, x_2) участва в оптималния хамилтонов цикъл, да отстраним това ребро от цикъла. Ще получим верига, състояща се от $n-1$ ребра (напомняме, че ребрата в хамилтоновия цикъл са n), минаваща през всички върхове с начало x_1 и край x_2. Теглото L на минималното покриващо дърво е очевидно долна граница за теглото на тази верига. Следователно

$$L(SST) + c(x_1, x_2) \leq Optimum(TSP). \tag{10.4}$$

В общия случай обаче може да не ни е известно никое от участващите в оптималния хамилтонов цикъл ребра, което прави невъзможно на практика определянето на цитираната горе долна граница. В [53] обаче е доказано, че най-дългото ребро в цикъла е с дължина не по-малка от $\max_{x_i}\{c(x_i, s)\}$, където с s е означен вторият най-близък връх до върха x_i. По този начин получаваме следната долна граница за TSP:

$$L + \max_{x_i}\{c(x_i, s)\} \leq Optimum(TSP). \tag{10.5}$$

РЕЛАЦИИ МЕЖДУ TSP, AP И SST. Нека G е неориентиран граф. Да означим с $G(TSP)$ покриващия подграф, състоящ се от ребрата на графа G, участващи в оптималния хамилтонов цикъл. Да означим с $G(AP)$ графа, включващ всички върхове и ребрата, участващи в оптималното решение на задачата за назначенията. Аналогично с $G(SST)$ графа, състоящ се от ребрата, участващи в оптималното решение на задачата SST.

Обърнете внимание на следното. Графът $G(TSP)$ притежава свойствата:

(1) Графът е свързан.

(2) Степента на всеки връх е 2.

Графът $G(AP)$ притежава свойството (2), но не винаги притежава свойство (1). Следователно, ако за решението на задачата за назначенията въведем изискването да се изпълнява свойство (1), това решение ще бъде решение на TSP, т.е. на задачата за търговския пътник.

Обратно, за графа $G(SST)$ свойство (1) е изпълнено, но той може да не притежава свойство (2). Следователно, ако за SST поискаме да се изпълнява и свойство (2) — с изключение на два крайни върха (примерно x_1 и x_2), които трябва да имат

степен 1, то минималното покриващо дърво ще бъде веригата с минимално тегло, минаваща през всички n върха. Ако освен това реброто (x_1, x_2) участва в оптималния хамилтонов цикъл, то ребрата на минималното покриващо дърво плюс реброто (x_1, x_2) дават решение на TSP.

От направените коментари е ясно, че са възможни следните два метода за решаване на TSP:

А) Използва се решението на задачата за назначенията AP (за което е изпълнено свойство (2)) и се опитваме да "подчиним" това решение на свойство (1).

Б) Използва се решението на задачата SST за минимално покриващо дърво (за което е изпълнено свойство (1)) и се опитваме да удовлетворим и свойство (2).

Направените бележки се отнасяха за графи със симетрична матрица на теглата. При ориентирани графи въведохме понятието ориентирано дърво (вж. параграф 2.2) аналог на понятието дърво. В този смисъл всичко, което казахме за връзката между TSP и минималното покриващо дърво в неориентирани графи, има точен еквивалент, отнасящ се до връзката между TSP и минималното покриващо ориентирано дърво в случая на ориентирани графи.

НАМИРАНЕ НА МИНИМАЛНИ ХАМИЛТОНОВИ ВЕРИГИ ЧРЕЗ SST *[2].* Нека T е покриващо дърво в графа $G = (X, A)$, а d_i^T е степента на върха x_i в дървото T. Отклонението на T от хамилтоновата верига може да се определи по следните два начина:

$$\varepsilon_T = \sum_{d_i^T > 2} (d_i^T - 2) \tag{10.6}$$

или

$$\varepsilon_T = \sum_{i=1}^{n} |d_i^T - 2| - 2. \tag{10.7}$$

Очевидно в (10.6) отклонението (близостта) се отчита само по върховете, за които $d_i^T > 2$, докато в (10.7) се взимат под внимание и висящите върхове. При хамилтонова верига $\varepsilon_T = 0$, поради което колкото по-голямо е ε_T, толкова повече дървото T се различава от хамилтоновата верига.

Да допуснем, че сме намерили минимално покриващо дърво T^* в графа G, такова че степените на всички върхове са ≤ 2. С други думи $d_i^T = 1$ или 2 за всички върхове i (дървото няма изолирани върхове). Ако с p означим броя на върховете от степен 1, тогава $n-p$ върха ще имат степен 2. Тогава за броя на ребрата в дървото ще получим

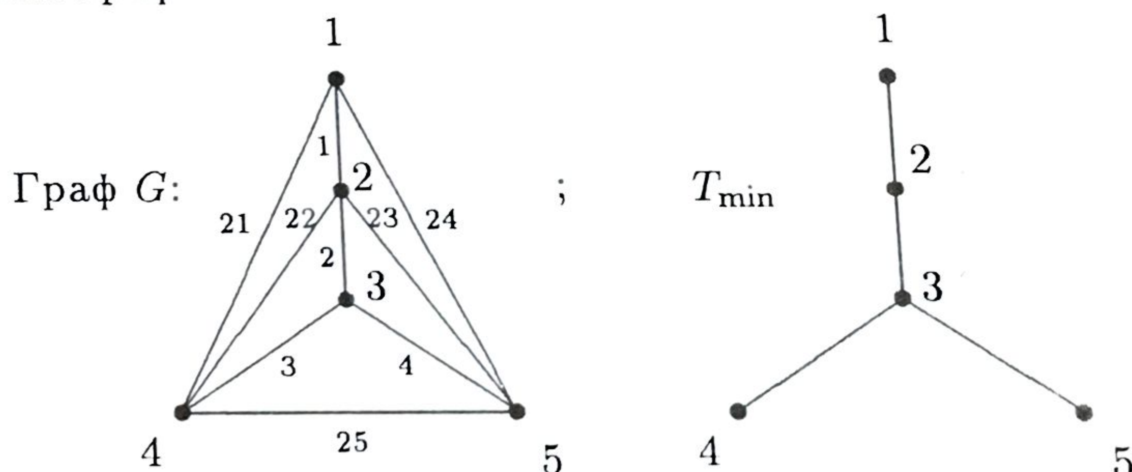
$$n - 1 = \frac{1}{2}\sum_{i=1}^{n} d_i^T = \frac{1}{2}(p + 2(n-p)) = n - \frac{p}{2} \Longrightarrow$$

$$n - 1 = n - \frac{p}{2} \Longrightarrow p = 2,$$

т.е. точно два върха са от степен 1, а $n-2$ върха са от степен 2, което означава, че T^* се явява хамилтонова верига.

Ще илюстрираме казаното със следния пример.

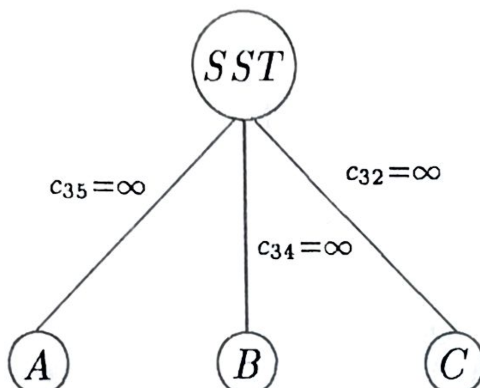
ПРИМЕР 10.1. Да се намери минимална хамилтонова верига в следния граф.



С алгоритъма за намиране на минимално покриващо дърво (вж. параграф 2.2) се намира решение $T_{\min}$ на SST със сумарно тегло 10.

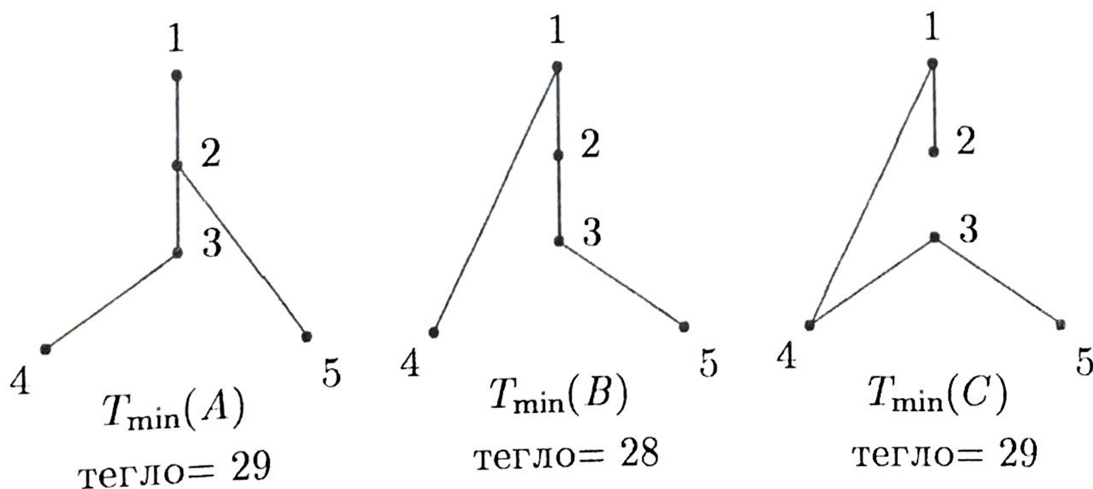
От чертежа горе вдясно се вижда, че в $T_{\min}$ $d_3^T = 3$, т.е. $T_{\min}$ не е хамилтонова верига, защото има връх със степен > 2. А на нас ни е необходимо да намерим такова минимално покриващо дърво T, за което степените на върховете в дървото са ≤ 2, което гарантира T да бъде хамилтонова верига. Следователно в оптималната хамилтонова верига трябва да отсъства поне едно от ребрата $(3,5)$, $(3,4)$ или $(3,2)$.

С други думи, решението на изходната задача се явява решение на поне една от следните три подзадачи, изобразени като върхове в дървото на решенията, дадено по-долу:

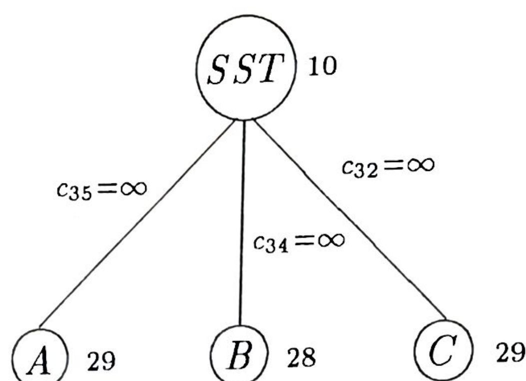


Върховете A, B и C са съответни на задачи със същите матрици на теглата като първоначалната задача, с тази разлика, че на указаните ребра е дадено тегло ∞, което гарантира изключването на съответното ребро от решението.

Да намерим сега минималните покриващи дървета $T_{\min}(A)$, $T_{\min}(B)$ и $T_{\min}(C)$, съответни на подзадачите A, B и C. Резултатите са дадени по-долу.



Следователно за дървото на решенията получаваме:



Долната граница на нашата задача е равна на най-малкото от теглата на покриващите дървета $T_{\min}(A)$, $T_{\min}(B)$ и $T_{\min}(C)$, т.е. долната граница е 28. Тъй като само $T_{\min}(B)$ и $T_{\min}(C)$ се явяват хамилтонови вериги с тегла съответно 28 и 29, то $T_{\min}(B)$ се явява оптималната хамилтонова верига.

Обърнете внимание, че във възела A по-нататъшни разклонения не е нужно да се правят, тъй като дори да получим (вследствие разклоняване на A) хамилтонова верига, нейното тегло няма да бъде по-малко от 29.

Ако в горното дърво решенията на подзадачите, съответни на върховете A, B и C не са хамилтонови вериги, избираме решението с минимално тегло за долна граница L и продължаваме да разклоняваме тази подзадача по същия начин. По-общо, след всяко разклоняване за долна граница L се избира теглото на онзи висящ връх (връх, в който не е направено разклоняване), който е с минимално тегло.

В много практически задачи обаче, не само се търси оптимална хамилтонова верига, а оптимална хамилтонова верига с фиксирани крайни върхове x_1 и x_2. Решаването на тази задача се реализира с гореописания метод, наречен "branch-and-bound", но след малка модификация, основаваща се на:

▷ ТЕОРЕМА 10.3. [2] *Нека $C = [c_{ij}]$ е матрицата на теглата (на ребрата) в графа G и M е достатъчно голямо положително число (M е по-голямо от теглото на всяка хамилтонова верига).*

Нека модифицираме матрицата на теглата C по следния начин до C'

$$C': \quad \left.\begin{aligned} c'_{j1} &= c_{j1} + M, \\ c'_{1j} &= c_{1j} + M, \\ c'_{2j} &= c_{2j} + M, \\ c'_{j2} &= c_{j2} + M, \end{aligned}\right\} \quad (\text{за всяко } x_j \neq x_1 \text{ или } x_2)$$

$$c'_{ij} = c_{ij} + 2M, \qquad (\text{за всяко } x_i \text{ и } x_j = x_1 \text{ или } x_2)$$

$$c'_{ij} = c_{ij}, \qquad (\text{за всяко } x_i,\ x_j \neq x_1 \text{ или } x_2).$$

Тогава хамилтоновата верига с минимално тегло при матрица на теглата C' е минимална хамилтонова верига с крайни върхове x_1 и x_2 при матрица на теглата C.

Доказателство: Възможни са и то само следните видове хамилтонови вериги:

а) x_1 и x_2 не са крайни върхове на веригата;

б) само следният от върховете x_1 и x_2 се явява краен връх на веригата;

в) x_1 и x_2 са крайни върхове на веригата.

Теглото на хамилтоновата верига при матрица на теглата C' е по-голямо от теглото на същата верига при матрица на теглата C с:

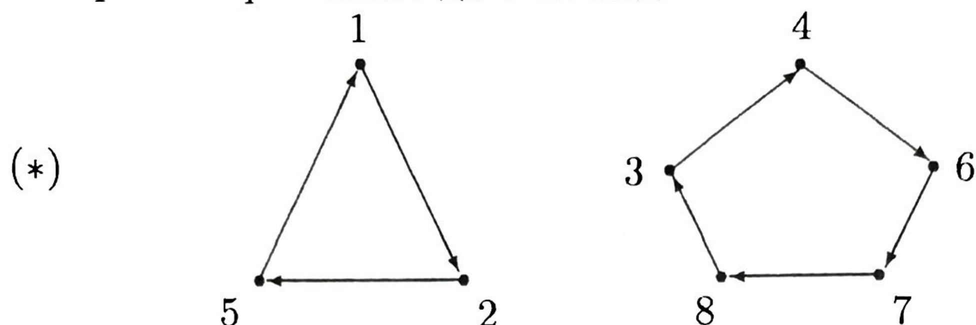
$4M$ — за вериги от тип а);
$3M$ — за вериги от тип б);
$2M$ — за вериги от тип в).

Тъй като M е достатъчно голямо (по-голямо от дължината на всяка хамилтонова верига), то теглото (при C') на най-дългата хамилтонова верига от тип в) е по-малко от теглото на минималната хамилтонова верига от тип б), а теглото на най-дългата хамилтонова верига от тип б) е по-малко от теглото на минималната верига от тип а). Следователно минималната хамилтонова верига при тегла C' дава хамилтонова верига от тип в). С това теоремата е доказана. ◁

3. Branch-and-bound алгоритми за TSP

РЕШАВАНЕ НА TSP *ЧРЕЗ* AP. Както вече изяснихме, решението на задачата за назначенията AP с матрица на теглата

$C = [c_{ij}]$, $c_{ii} = \infty$ за $\forall i$, в общия случай се състои определен брой непресичащи се цикли (ако цикълът е един, той се явява оптималния хамилтонов цикъл). Например решението на задачата AP при 8 върха може да е от вида

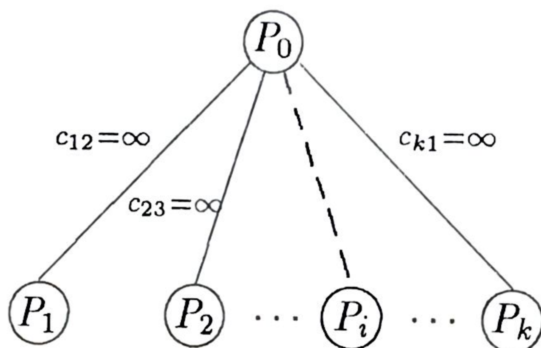


(т.е. първият работник върши втората работа, вторият работник — петата работа, третият работник — четвъртата работа и т.н.).

Ясно е, че ние трябва да изключим даденото решение и всяко друго решение, състоящо се от повече от един цикъл, без разбира се да изгубим решението на TSP. Това може да се направи така:

МЕТОД НА ПРОСТОТО РАЗКЛОНЯВАНЕ (SIMPLE BRANCHING RULE). Този "branch-and-bound"-алгоритъм се състои в следното: ако решението на задачата AP се състои от един цикъл, този цикъл е оптималното решение на TSP.

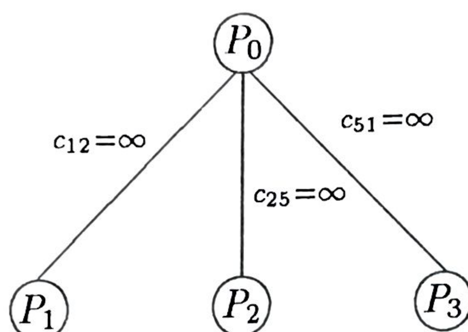
Нека решението на AP съдържа цикъл $(x_1,\ x_2,\ \ldots,\ x_k,\ x_1)$, който не е хамилтонов. Отстраняването на този цикъл (и всички решения, които го съдържат) от по-нататъшно разглеждане, се осъществява като поне една от дъгите (x_1, x_2), (x_2, x_3), ..., (x_k, x_1) се отстрани от решението. Отстраняването се реазилизира на практика като изходната задача с матрица на теглата $[c_{ij}]$ се разбие на k подзадачи $P_1,\ P_2,\ \ldots,\ P_k$, както е показано по-долу.



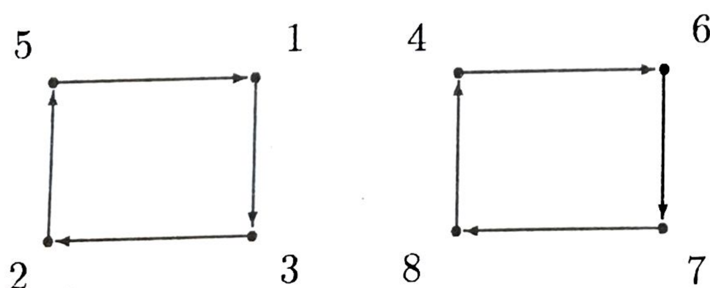
Ясно е, че решението на задачата P_0, несъдържащо цикъла $(x_1,\ x_2,\ \ldots,\ x_k,\ x_1)$, се явява решение на поне една от подзадачи-

те P_1, P_2, ..., P_k. С други думи, оптималното решение на TSP е решение на една или няколко от тези подзадачи.

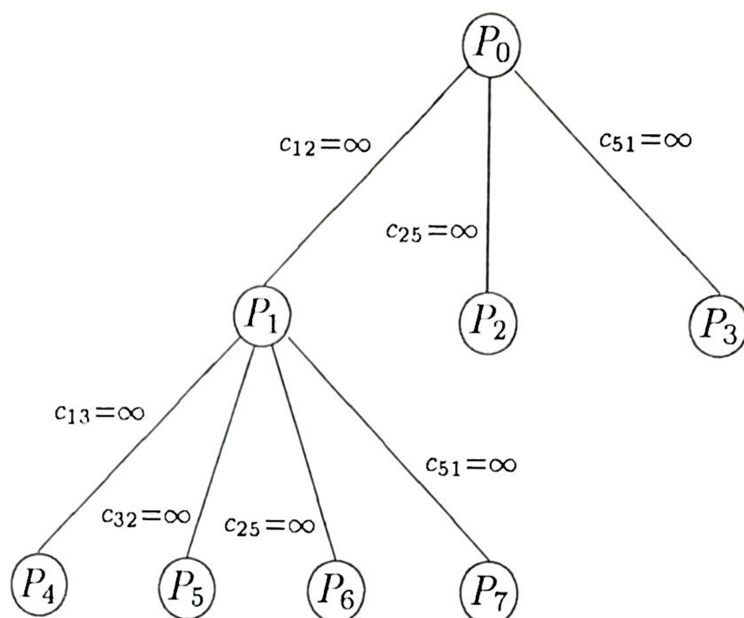
Нека да изключим цикъла с дължина 3 от (*). Ще получим следното дърво на решението:



Ако решим всяка от задачите P_1, P_2 и P_3 като задача за назначенията, да означим съответните тегла на решенията със C_1, C_2 и C_3. Тъй като C_i се явява долна граница за TSP в P_i, $i = 1, 2, 3$, то $L = \min\{C_1, C_2, C_3\}$ се явява долна граница на теглото на решението за изходната TSP задача. За определеност, без да ограничаваме общността на разсъжденията, да приемем, че $C_1 \leq C_2 \leq C_3$, т.е. $L = C_1$. Ако решението на задачата P_1 е хамилтонов цикъл, то това решение ще бъде оптимално-то решение на началната TSP задача. В противен случай, нека например решението на задачата за назначенията P_1 е следното:



Изключваме цикъла (1, 3, 2, 5, 1) и отново решаваме подзадачите P_4, P_5, P_6 и P_7, показани долу:



От чертежа се вижда, че задачата P_4 е задача, за която в матрицата на теглата на P_1 елементът $c_{13} = \infty$. Новата долна граница се определя като

$$L = \min\{C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ C_7\},$$

където със C сме означили теглата на решенията на съответните задачи.

Да допуснем, че $L = C_2$. Тогава, ако решението на задачата P_2 е хамилтонов цикъл, това решение ще бъде решение на първоначалната TSP задача. В противен случай се извършва разклоняване във върха P_2 по гореописания начин (както разклонявахме върха P_1). Процедурата на разклоняване и определяне на долна граница се извършва, докато решението на задача с текущо тегло L стане хамилтонов цикъл. Това е решението, т.е. оптималният хамилтонов цикъл за изходната задача.

МЕТОД НА ИЗКЛЮЧВАЩОТО РАЗКЛОНЯВАНЕ. Както изяснихме в параграф 1.8, твърде полезно е разбиването на задачата P_i на подзадачи да се осъществи така, че всяко допустимо решение на задачата P_i да бъде решение на една и само една от нейните подзадачи. За да се реализира това, се прилагат следните правила за разклоняване, с цел отстраняване на цикъла $(x_1,\ x_2,\ \ldots,\ x_k,\ x_1)$, водещи до взаимно изключващи се подзадачи:

За задачата P_1 полагаме $c(x_1, x_2) = \infty$.

За задачата P_2 полагаме $c(x_1, x_2) = -M$ и $c(x_2, x_3) = \infty$.

За задачата P_3 полагаме $c(x_1, x_2) = c(x_2, x_3) = -M$ и $c(x_3, x_4) = \infty$.

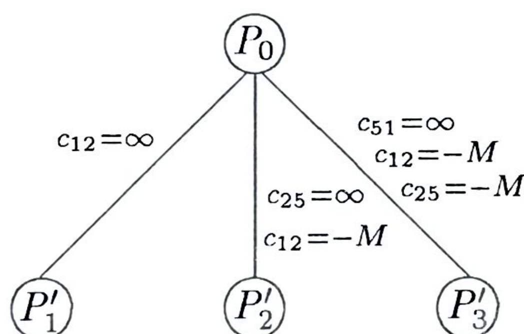
....

За задачата P_k полагаме $c(x_1, x_2) = c(x_2, x_3) = ... = c(x_{k-1}, x_k) = -M$ и $c(x_k, x_1) = \infty$,

където $(-M)$ е достатъчно голямо отрицателно число, което гарантира дъгите с тегло $(-M)$ да участват в оптималното решение.

Това правило на разклоняване води до взаимно изключващи се подзадачи, тъй като всеки две подзадачи имат поне една дъга, изключена от решението на едната, но задължително влизаща в решението на другата.

При този метод на изключващо разклоняване задачата P_0 от последния чертеж ще се разбие на следните три подзадачи.



В [2] е даден още един тип разклоняване, който ние няма да разглеждаме.

ЕВРИСТИЧЕН АЛГОРИТЪМ ЗА TSP. Разгледаните досега методи за решаване на TSP бяха от т.нар. клас "точни методи" — методи, които гарантират получаване на оптимално решение. За съжаление алгоритмите, реализиращи тези точни методи, в много случаи са неефективни и неприложими на практика. Ето защо, понякога за практически нужди е полезно да се намери решение на даден проблем, което не винаги се явява оптималното. При това с помощта на долната граница за теглото на оптималното решение може да се оцени колко е отклонението (близостта) на полученото решение от (до) оптималното. Този тип алгоритми, даващи решение на проблема, без гаранция за оптималност, се наричат *евристични алгоритми (heuristic algorithms)*.

Един такъв семпъл евристичен алгоритъм, решаващ TSP е следният:

Вземете произволен хамилтонов цикъл. Означете с $(x_1, x_2, ..., x_n)$ последователността, в която се обхождат върховете на графа G.

За $i = 1, 2, ..., n-1$ и $j = i+1, ..., n$ проверете дали се получава хамилтонов цикъл с по-малка дължина при смяна на местата на x_i и x_j в горната последователност на обхождане. Ако това е така, отразете тази смяна на местата в последователността на обхождане.

Очевидно процесът на смяна в реда на обхождането е краен, тъй като са краен брой различните начини на обхождане на n върха и освен това всеки път се формира нова, различна от предишните, последователност на обхождане.

За съжаление в предложения алгоритъм изборът на начален хамилтонов цикъл не влияе върху "оптималността" на получаваното решение. Възможно е да тръгнете от първоначален хамилтонов цикъл с "много лоша" дължина и да стигнете до цикъл с "много добра" дължина, дори до оптималния хамилтонов цикъл. Както и обратното — стартирате от цикъл с не толкова лоша дължина, но получавате решение, съществено различаващо се от истинското оптимално решение. Възможно е да тръгнете от начален хамилтонов цикъл и на всяка смяна на реда на обхождане да не съответства нов хамилтонов цикъл. С други думи, алгоритъмът не намира по-добър от началния хамилтонов цикъл.

Литература

[1] Minieka, E., *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, Inc., New York and Basel, 1978 (Майника, Э. *Алгоритмы оптимизации на сетях и графах*, М., "Мир", 1981).

[2] Christofides, N., *Graph Theory. An Algorithmic approach*, Academic Press Inc (London) Ltd. 1975, 1977 (Кристофидес, Н. *Теория графов. Алгоритмический подход*, М., "Мир" 1978.

[3] Swamy, M., K. Thulasirman, *Graphs, Networks and Algorithms*, John Wiley & Sons, 1981 (Свами М., К. Тхуласирман. *Графы, сети и алгоритмы*, М., "Мир", 1984).

[4] Зайченко, Ю. П., *Исследование операций*, Киев, "Высша школа", 1988.

[5] Ore, O., *Graphs and Their Uses*, Random House new Mathematical Library, Random House, New York, 1963 (Оре, О. *Графы и их применение*, М., "Мир",1965).

[6] Papadimitriou, Ch., K. Steigeitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc Englewood Cliffs, New Jersey 1982 (Пападимитриу, Х., К. Стайглиц. *Комбинаторная оптимизация, Алгоритмы и сложность*, М., "Мир", 1985).

[7] Мухачева, Э., Г. Рубинштейн, *Математическое програмирование*, Новосибирск, "Наука", Сибирское отдел, 1987.

[8] Harary, F., *Graph Theory*, Addison-Wesley, Reading, 1969 (Харари, Ф. *Теория графов*, М., "Мир", 1973).

[9] Фудзисава, Т., Т. Касами, *Математика для радиоинженеров. Теория дискретных структур*, М., "Радио и связь", 1984.

[10] Ford, L. R., D. R. Fulkerson, *Flows in Networks*, Princeton Press, Princeton 1962 (Форд, Л. Р., Д. Р. Фалкерсон, *Потоки в сетях*, М., "Мир", 1966).

[11] Dantzig, G. B., *Linear Programming and Extensions*, Princeton Press, Princeton, 1963 *(Данциг, Дж. Линейно програмиране и неговото приложение,*превод от руски, Държ. изд. - Варна, 1973).

[12] Йорданов, Й., *Количествени методи в управлението*, СУ "Св. Кл. Охридски", Нац. център за дистанц. обучение, София, 1997.

[13] Иванчев, Д., Г. Неглер, *Мрежово оптимиране*, Техн. университет, София, 1993.

[14] Edmonds, J., *Optimum Branchings*, Mathematics of the Decision Sciences, Lectures in Applied Mathematics, Vol. 2, AMS., pp. 346-361, 1968.

[15] Dijkstra, E. W., *A Note on Two Problems in Connection with Graphs*, Numer. Math., 1, pp. 269-271, 1955.

[16] Ford, L. R., *Network Flow Theory*, Raud Corporation Report, p. 923, 1946.

[17] Floyd, R. Z., *Algorithm 97, Shortest Path*, Comm. ACM, 5, p. 345, 1962.

[18] Dantzig, G. B., *All Shortest Routes in a Graph, Theory of Graphs*, International Symposium, Rome, Gordon and Breach, New York, pp. 91-92, 1966.

[19] Hu, T. C., *Integer Programming and Network Flows*, Addison-Wesley, Reading, Massachusetts.

[20] Bellman, R., R. Kalaba, *On k^{-th}-Best Policies*, J. of SIAM, 8, p. 582, 1960.

[21] Hoffman, W., R. Pavley, *A Method for the Solution of the N^{-th} Best Path Problem*, J. of ACM, 6, p. 506, 1959.

[22] Yen, J. Y., *On the Efficiencies of Algorithms for Detecting Negative Loops in Networks*, Santa Clara Business Review, p. 52, 1971.

[23] Гэри, М., Д. Джонсон, *Вычислительные машины и труднорешеемые задачи*, М., Мир, 1982.

[24] Karp, R. M., *On the Computational Complexity of Combinatorial Problems Networks*, 5, pp. 45-68, 1975.

[25] Knuth, D. E., *The Art Computer Programming, vol.3, Sorting and Searching*, Addison-Wesley, Reading, 1973 (Кнут, Д. Е. *Искусство програмирования для ЭВМ. Т. 3. Сортировка и поиск.* М., Мир, 1979).

[26] Lawler, E. L., *The Complexity of Combinatorial Computations: A Survey*, Proc. 1971, Politechnic Institute of Brooklyn Symposium on Computers and Automata, 1971.

[27] Williams, T. A., G. P. White, *A Note on Yen's Algorithms for Finding the Lenght of All Shortest Paths in N-Node Nonnegative Distance Networks*, J. ACM, 20, No 3, pp. 389-390, 1973.

[28] Yen, J. Y., *Finding the Lenghts of All Shortest Paths in N-Node Nonnegative Distance Complete Networks Using 1/2 N^3 Additions and N^3 Comparisons*, J. ACM, 19, No 3, pp. 423-424, 1973.

[29] Jonson, E. L., *On Shortest Paths and Sorting*, Proc. of Annual Couf. of ACM, Boston, p. 510, 1972.

[30] Dreyfus, S. E., *An Appraisal of Some Shortest Paths Algorithms*, Ops. Research, 17, pp. 395-412, 1969.

[31] Edmonds, J. Karp R. M., *Theoretical Improvements in Algorithm Efficiency for Network Flow Problems*, J. ACM, vol. 19, N. 2, pp. 218-264, 1972.

[32] Eisner, H., *A Generalized Network Approach to the Planning and Scheduling of a Research Project*, ORSA, 10, pp. 115-125, 1962.

[33] Elmaghraby, S., *An Algebra for the Analysis of Generalized Activity Networks*, Mgmt. Sci., 10, N 3, pp. 494-514, 1964.

[34] Pritsker, A. B., *Modeling and Analysis Using Q-GERT Networks*, Halsted Press, New York, 1977.

[35] Hakimi, S. L., *Optimum Locations of Switching Center and the Absolute Centers ant Medians of a Graph*, ORS, 12, pp. 450-459, 1964.

[36] Христов, Г., Р. Калтинска, *Математическо оптимиране, част първа*, Н. И., С., 1972.

[37] Христов, Г., Р. Калтинска, *Числени методи 10*, Народна просвета, С., 1978.

[38] Кънчев, К., Н. Стойнова, К. Бонев, *Линейна алгебра и математическо програмиране*, Варна, 1971.

[39] Кънчев, К., К. Бонев и др., *Математическо програмиране*, "Г. Бакалов", Варна, 1975.

[40] Кендеров, П., Г. Христов, Ас. Дончев, *Математическо оптимиране*, Унив. изд. "Кл. Охридски", С., 1989.

[41] Христов, Г. и др., *Ръководство за решаване на задачи по математическо оптимиране*, Унив. изд. "Св. Кл. Охридски", С, 1989.

[42] Чимев, К., Ив. Мирчев, *Висша математика*, Унив. изд. "Н. Рилски", Благоевград, 1999.

[43] Данко, П. Е. и др., *Высшая математика в упражнениях и задачах*, Москва, "Высшая школа", 1980.

[44] Липский, В., *Комбинаторика для програмистов*, Изд. "Мир", Moskva, 1988.

[45] Келеведжиев, Е., *Динамично оптимиране (ръководство за решаване на задачи по програмиране)*, Изд. "Анубис", С., 2001.

[46] Edmonds, J., *Maximum Matching and Polyhedra with 0-1 Vertices*, J. Res, N. B. S. vol 69B, No 1, 2, pp. 125-130, 1965.

[47] Evans, J., E. Minieka, *Optimization Algorithms for Networks and graphs*, Marcel D., Inc., 2-nd ed., rev, 1992.

[48] Диниц, Е. А., *Алгоритм Решения задачи о максимальном потоке в сети со степенной оценкой*, Докл. Акад. наук СССР, Серия Мат., Физ., 1970, 194, 4, с.754-757.

[49] Hopcroft, J., R. M. Karp, *An $n^{5/2}$ algorithm for maximum Matchins in bipartite graphs*, SIAM J. Comput., 1973, 2.

[50] Even, S., O. Kariv, *An $n^{5/2}$ algorithms for maximums Matching in general graphs*, Proc. 16-th Annual Symp. on Foundations of Comp. Sc. IEEE, 1975, p. 100-112.

[51] Edmonds, J., Jonsin Ellis, *Matching: A Well Solved Class of Integer, Linear Programs, Combinatorial Structures and Their Applications*, Gordon and Breach, New York, p. 89-92, 1970.
Marcel D., Inc., 2-nd ed., rev, 1992.

[52] White, L. J., *A Parametric Study of Matching and Coverings in Weighted Graphs*, Ph. D. Thesis, University of Michigan, 1967.
Marcel D., Inc., 2-nd ed., rev, 1992.

[53] Eilon, S., Watson - Gandy C. D. T., N. Christofides, *Distribution Management: Mathematical Modelling and PracticaL Analysis*, Griffin, London, 1971.

[54] Мирчев, Ив., *Оптимиране. Алгоритмичен подход*, Унив. изд. "Н. Рилски", Благоевград, 1999.

Указател

Абсолютен център на граф 353
абсолютна медиана в граф 354
абстрактни събития 329
алгебричен поток 283
алгоритъм за задачи от тип TSP 403, 417
алгоритъм за k-ти по дължина пътища (Йен) 260
алгоритъм за максимално по мощност сдвояване 361, 380
алгоритъм за максимално по тежест сдвояване 362, 381, 385
алгоритъм за минимално по тежест покритие 388
алгоритъм за намиране на най-ранните срокове (най-късните срокове) 334
алгоритъм за построяване на покриващо дърво 224
алгоритъм за построяване на минимално и максимално покриващо дърво 228
алгоритъм за решаване на CPP 395
алгоритъм за решаване на TSP чрез AP 413
алгоритъм за съвършени сдвоявания с минимално тегло 362
алгоритъм за търсене на поток с минимална цена 305
алгоритъм за топологическа сортировка 331, 332
алгоритъм за търсене на увеличаваща потока верига 285
алгоритъм на Данциг за НКП 249
алгоритъм на Дийкстра за НКП 243
алгоритъм на дефекта 306, 317
алгоритъм на Едмондс за МОЛ 233
алгоритъм на Едмондс и Карп за максимален поток 299
алгоритъм на Йен 260
алгоритъм на симплекс-метода 201
алгоритъм на Флойд за НКП 249
алгоритъм на Форд за НКП 247
алгоритъм на Форд-Фалкерсон за максимален поток 289
алгоритъм с експоненциална сложност 271
алгоритъм с полиномна сложност 270
алтернативен вход 346
алтернативна верига 92, 373
алтернативно дърво 374
антисиметричен граф 14
аритметичен поток 283
аугментална (усилваща) верига 92, 374, 382
аугментално дърво 375
ацикличен граф 14

База 59
базис на опорен план 189
базисен вид на система 190
базисни вектори 189
базисни променливи 189
базисни (фундаментални) цикли 38
базисно представяне на система 190
баща за връх 158
безкрайни (неограничени) области 131
биполярен (двухроматичен) граф 15
блок 25
букет (цвят) 224, 375
"branch-and-bound" алгоритми 413-417

Величини на дефекта 320
верига 10

вектор на ограниченията, на условията, на неизвестните 187
вектори на инцидентност 47
верижно сдвояване 395
вероятностен алгоритъм 277
вероятностен изход 346
висящи върхове 8
време за пренос през дъга 324
времева сложност 269
входяща дъга 8
външен (terminal) метод 256
външни върхове 374
външно устойчиво множество 105
върхова свързаност на граф 75
върхове на граф 6
върхово k-оцветяване 141
върхово непресичащи се пътища 81
върхово покритие в граф 104
вътрешен (вграден) метод (tentative method) 253
вътрешни върхове 374
вътрешно произведение на върхове 125
вътрешно устойчиво множество 103

Главен (абсолютен) център 354
главна медиана на граф 356
главни детерминанти 50
гора (лес) 21
горна граница 168
граница на област 131
граф 6
граф на Петерсен 137
гъстота (плътност) 103

Двойнствени (дуални) графи 138
детерминиран изход 346
дефект (цикломатично число) 38
дефицит на граф 86
дефицит на множество 85
дизюнктивен вход 346
динамичен поток 324
динамично оптимиране 177
Дийкстра 243
дискретно оптимиране 178
долна граница 168
долна граница на TSP чрез AP 407
долна граница на TSP чрез SST 408
доминиращо множество от върхове 105
допълнение на граф 19
допълнителна променлива 194
дуална (двойнствена, спрегната) задача 212
дъги на дъга 222
дъги на граф 6
дълбочина на връх 159
дължина (мощност) на верига 10
дължини 242
дължина на контура 11
дължина на път 11
дължина на цикъл 11
дърво 21

Евристични алгоритми (heuristic algorithms) 278, 417
евристичен алгоритъм за TSP 417
единици (на потока) 282
еднокомпонентен (свързан) граф 15
еднороден (регулярен) граф 14
едностранна компонента на граф 42
едностранно свързан граф 42

Жаден алгоритъм 142-144

Задача за играчките 84
задача за китайския пощальон (The Chinese Postman Problem (CPP) 115, 393
задача за клюката 223

задача за назначенията 85, 362
задача за оптимално разпределение на ресурси 145, 179
задача за сватбите 83
задача за снегорина 116
задача за съставяне на графици 144
задача за търговския пътник 119, 266, 403
задача на линейното оптимиране 182
задача с експоненциална сложност 270
задача с полиномна сложност 270
задачи за избор на маршрут 181
задачи за масово обслужване 180
задачи за мрежово планиране и управление 180
задачи за разположение на обекти 181
задачи за ремонт и подмяна на оборудване 179
задачи за управление на запасите 179

Изкуствен базис, M-метод 196
изкуствена променлива 194
изолирани върхове 8
изоморфни графи 20
изоморфни модели 176
изроден план 188
изродена задача 188
източник 282
изходяща дъга 8
инцидентни връх и дъга (ребро) 7
изчислителен вариант на задача 272
изчислителна сложност 153

Канонична задача (форма, запис) 183, 187
капацитет (пропускателна способност) на дъга 282
карта 131
класическа транспортна задача 221
класове на сложност - $\mathcal{NP}$, $\mathcal{NPC}$, $\mathcal{NPI}$, $\mathcal{NPH}$... 273-278
клика 103
клони на дърво 22
ко-дърво 22
коефициент на усилване на дъга 265
коефициент на усилване на път 265
конюнктивен вход 346
корен на търсенето 158
корен на граф 59
корен на ориентирано дърво 21
корен на отклонението 260
комбинирани задачи 181
компонента 41
компоненти на граф 15
компромисно решение 178
кондензация на граф 61
конкретна (индивидуална) задача 267
контур 11
коцикломатично число (ранг) 38
краен връх на път 11
краен цвят 376
край на верига (краен връх на верига) 10
крайни области 131
критерий за оптималност 197, 216
критичен път 328, 340
критична операция 340
k-оцветим граф 141
k-ребрено-свързан граф 75
k-свързан граф 76
k-ти по дължина път 260
k-хроматичен граф 142

Лексикографски динамични потоци 327
лес (гора) 21

линейно оптимиране 177
линия 89

Максимален динамичен поток 324
максимален (минимален) ориентиран лес 232, 240
максимален подграф 19
максимална пропускателна способност (максимален капацитет) 317
максимална пропускателна способност на дъга 282
максимален поток 284
максимален пълен подграф (клика) 103
максимално независимо множество 103
максимално по мощност съчетание 360, 373
максимално по тежест (тегло) съчетание 360
максимално подмножество 19
максимално покриващо дърво 228
максимално покриващо ориентирано дърво 240
максимално сдвояване 83
максимално увеличение на потока по веригата 287
маркиран (разгледан) връх 158
маркирано (разгледано) ребро 158
маркиращо число 243
матрица на достижимост R 53
матрица на инцидентност A_I 47
матрица на контрадостижимост Q 53
матрица на ограничена достижимост и контрадостижимост 57
матрица на ограниченията (условията) 187
матрица на разрезите S 69
матрица на степените D 62
матрица на съседство B 52
матрица на теглата 242
матрица на фундаменталните разрези S_f 71
матрица на фундаменталните цикли C_f 67
матрица на циклите C 67
математическо моделиране 181
медиани на граф 350, 354
метод за решаване на TSP 406
метод на изключващото разклоняване (disjoint branching rule) 416
метод на изкуствения базис 196
метод на обективно обусловените оценки 193
метод на простото разклоняване (simple branching rule) 414
минимален коефициент на усилване 265
минимален подграф 19
минимален поток 305
минимален разрез 296
минимално свързан граф 43
минимаксна модификация 120
минимаксни задачи за разположение 350
минимална пропускателна способност (минимален капацитет) 317
минимално върхово покритие 105
минимално по мощност покритие 99, 360
минимално (по мощност) ребрено покритие 98
минимално по тежест (тегло) покритие 99, 360, 388
минимално подмножество 19
минимално покриващо дърво 227
минимално покриващо ориентирано дърво 240
минимално свързан неориентиран граф 44

минисумарен характер 120
минисумарни задачи за разположение 350
многокритериални, многоцелеви задачи 178
множество от всички индивидуални задачи 267
множество от съществени върхове относно върховете v_i и v_j 57
модел, моделиране 176
модифициран симплекс-метод 210
модифицирана матрица на съседство 125
мост 78
мрежа 13, 222
мрежови график 329
мрежово планиране 328
мултиграфи 7
мярка за външна устойчивост 105
мярка за доминираност 105
мярка за независимост (мярка за вътрешна устойчивост) 103
M-метод (задача) 196, 207

Намаляващи дъги 284
напълно сканиран (използван) връх 159
наредени двойки 6
наситен връх 83
наситени върхове 92
наситени (тъмни) върхове 374
наследник (потомък на връх) 163
най-кратък път (НКП) 242
най-късен срок 333
най-надежден път 264
най-ранен срок 333
настъпило събитие 330
начален връх 158
начален връх на път 11
начален и краен връх на дъга 7
начална (изходна, пряка) задача 212
начало на верига (начален връх на верига) 10
независим резерв от време 339
независими пътища 81
независими ребра 83, 150
независимо множество 103
нелинейно оптимиране 177
ненаситени върхове 92
ненаситени (светли, експонирани) върхове 374
неориентиран граф 6
неориентиран дубликат, двойник 7
непосредствено предшестваща операция 329
неразделими свързани графи 24
несиметрични дуални задачи 215
несъществени върхове 57
неуплътнен връх 390
неутрална увеличаваща верига 382
нечетна компонента в граф 96
ниво на важност на цели 178
нулева точка 350
$\mathcal{NP}$- пълни задачи 169

Обединение на графи 20
обикновен симплекс-метод 196
области (страни) на равнинна карта 131
обобщени мрежови графици 345
обосновка на алгоритъма за намиране на максимално покриващо дърво 228
обратни дъги 286
обратно ребро 159
обща форма на дуалната задача 213
ограничена достижимост и контрадостижимост 57
ойлеров граф 110
ойлеров цикъл 110

опорен план 187
оптимален план (решение) 177
оптимизационен вариант на задача 272
ориентиран граф 6
ориентиран лес (гора) 22
ориентирано дърво 21
ориентирано дърво на най-кратките пътища 246
ортогоналност на A_I и C_f^t 72
ортогоналност на C_f и C_f^t 73
остатъчна последователност 63
остатъчни степени на върхове 144
отворена ойлерова верига 110
отклонение от път 260
отстраняване на връх 20
отстраняване на ребро 20
отъждествяване на два върха 20
оцветен връх 243
оцветяване на върхове 141
оцветяване на ребра 224
(обща) задача на търговския пътник (англ. Travelling Salesman Problem) 403

Паралелни дъги (ребра) 7
подграф 17
подграф на граф, породен от върхове 18
подграф на граф, породен от ребра 18
план 177
планарен граф 128, 129
планарно изображение 129
плътност (гъстота) 103
поглъщащи алгоритми 224
полустепен на вход на връх 8
полустепен на изход на връх 8
покриващ ориентиран лес 22, 239
покриващ ойлеров цикъл 110
покриващо ориентирано дърво 21, 22
покриващи множества 108
покриващо дърво 21
покриващо ориентирано дърво 22
покритие 360
покритие с максимална мощност и максимално тегло 99
покритие на R 108
полиномна трансформация на задача 275
последователни ребра 136
постоянно маркиращо число 243
поток 282
поток в дадена дъга 283
поток в мрежа 283
поток с минимална стойност 305
потомък (наследник) на връх 163
прави дъги 286
правилен многостен 133
правилно (върхово) k-оцветяване 141
правилно ребрено k-оцветяване на граф 149
празен връх 390
предшественик на връх 163
пренос с минимална цена 305
примка 7
проекти с минимална стойност 342
произволно-ойлеров граф 117
произволно-ойлерови графи относно връх v 117
променливи - статични, динамични, дискретни, непрекъснати, детерминирани, случайни 177
прост граф 12
пропускателна способност (капацитет) на дъга 282
прости верига, път, контур или цикъл 11
просто разрязващо множество 32
пълен биполярен граф 16

пълен граф 12
пълен резерв от време 338
пълен r-хроматичен граф 16
пълно сдвояване 84
път (ориентирана верига) 11
път с максимална дължина на най-късата дъга 263
път с минимална дължина на най-късата дъга (път с минимална пропускателна способност) 264

Равнинна карта 131
разбиване 108
разделящо множество 76
разделими свързани графи 24
разклонение на път 260
разклоняване (branching) 169
разпознавателен вариант на задача 272
разполагане на граф 129
разполагане на обекти 349
разпределителен метод 222
разрез на граф 33
разрешаващи възли 345
разрешимост на подзадача 167
разрязана (пресечена) матрица на инцидентност 48
разрязващо множество 32
разстояния в граф 350
разстояние от тип "връх-връх" (ВВ) 350
разстояние от тип "връх-дъга (ребро)" (ВД) 351
разстояние от тип "точка-връх" (ТВ) 351
ранг на граф 23, 38
ребра на граф 6
ребрен граф 121
ребрена свързаност 75
ребрено покритие 97
ребрено покритие с минимално тегло 98
ребрено k-оцветим граф 149
ребрено k-оцветяване на граф 149
ребрено k-хроматичен граф 150
ребрено хроматично число 150
ребро на дърво 158
регулярен (еднороден) граф 14
редуваща се (алтернативна) верига 92
релации между TSP, AP и SST 408
рефлексивен граф 14
решение (оптимален план) 177

Светли ребра 91, 374
свиване 20
свиване на цветове 376
свободен резерв от време 338
свободни променливи 182
свързан граф 13
свързан ориентиран граф 14
свързваща точка 24
сдвояване (съчетание по двойки) 83, 360
сечение на графи 20
силна свързаност 41
силна увеличаваща верига 382
силно свързан граф 41
силно свързани върхове 41
силно свързани компоненти на граф 42
символ на Ландау 153, 268
симетричен граф 14
симетрични дуални задачи 214
симплекс-метод 193-210
симплекс-метод с естествен базис 196
симплекс-метод с изкуствен базис 196
симплекс-таблици 202

син 163
система различни преставители на фамилията F (трансферзала) 88
скорост на нарастване на сложността на алгоритъм 270
скрити цени 212
слаба свързаност 41
слаба увеличаваща верига 382
сложност 268
сложност на алгоритъм, на задача 268
сложност по памет 269
смесен граф 8
собствен подграф 17
списък двойки съответни на ребра 156
списък на инцидентност 156
списък на ребрата 156
стандартна задача 183
степен на важност на целите 178
степен на връх 8
степен на достигане на цели 178
степен (тегло) на важност 178
стереографична проекция 129
сток (краен пункт) 282
стохастично оптимиране 178
страни (области) на равнина 131
събития 330
съвършени (покриващи) сдвоявания 84
съвършено сдвояване 84
съответна подматрица 49
съотносими върхове 163
съседни върхове 7
съседни дъги (ребра) 7
съседни области 131
съседни страни 131
s-граф 7
$(s-t)$-разрез 294

Тегло, дължина, стойност, цена на дъга 12, 222
тегло (дължина, стойност, цена) на път 13, 222
тегло на дървото 223
теорема на Берж 92
теорема на Бине-Коши 50
теорема на Куратовски 136
теорема на Кьониг - Еге-вари 89, 91, 107
теорема на Кьониг 87
теорема на Менгер 81
теорема на Менделсон и Далмедж 95
теорема на Ойлер 130
теорема на Тат 96
теорема на Хол 86
теореми за дуалност (двойнственост) 216-221
теорема за равновесието 220
топологическа сортировка 330
транзитивен граф 14
транспортна задача 221
трансферзала 88
тривиален граф 14
тъмни ребра 91, 374
търсене в дълбочина 158
търсене в ширина 164
търсене с връщане назад backtraking 143, 169
тясно място във верига 302

Унгарски алгоритъм (Кун) 366
унгарско дърво (hungarian tree) 378
унимодулярна матрица 49
уплътнен връх 390
увеличаваща (усилваща, аугментална) верига 92, 374, 382
увеличаващи (ненаситени) дъги 284

Фиктивна дъга 329

формула на Ойлер 130
фундаментална (базисна) матрица на разрязващите множества 71
фундаментални (базисни) цикли 38
фундаментални разрези относно покриващо дърво T 40
фундаментална теорема 217
f-наситена дъга 296
f-ненаситена дъга 296
f-нулева дъга 296
f-положителна дъга 296

Хамилтонов граф 118
хамилтонов цикъл 118
хамилтонова верига (път, маршрут) 118, 409
характеризация на Вагнер, Харари и Тат 137
характеризация на Куратовски-Понтрягин 136
хипотеза за четирите цвята 140
хомеоморфни графи 136
хомоморфни модели 176
хорда 22
хроматичен индекс 150
хроматични класове 15
хроматично число 142

Цвят (букет) 224, 375
целева функция 177
цена (стойност) на преместване на единица от поток по дъга 283
цена (стойност, разходи) 305
център на граф 353
циклично ребро 24, 27
цикломатична матрица (матрица на циклите) 67
цикломатично число на граф 23
цикъл 11
циркулации 318

Частен случай на задача 278
четна компонента в граф 96

$(0,1)$-матрици 89
2-хроматичен (биполярен) граф 15

Съдържание

Предговор **3**

1 Теория на графите. Алгоритмичен подход **5**

1.1. Основни понятия в теория на графите . . . 5
1.2. Видове графи. Подграф. Операции с графи. Дървета . . . 12
1.3. Свързаност. Покриващи дървета. Разрези. Цикли. Силна свързаност . . . 23
1. Свързаност . . . 23
2. Дървета . . . 26
3. Разрези и цикли . . . 32
4. Силна свързаност . . . 41
1.4. Матрично представяне на графи . . . 47
1. Матрица на инцидентност . . . 47
2. Матрица на съседство . . . 51
3. Матрици на достижимост . . . 53
4. Матрица (вектор) на степените . . . 62
5. Матрици на циклите и разрезите . . . 66
1.5. K-свързани графи. Сдвоявания. Покрития . . . 74
1. Ребрена и върхова k-свързаност . . . 74
2. Теорема на Менгер . . . 80
3. Съчетания по двойки (Сдвоявания) . . . 83
4. Сдвоявания в биполярни графи . . . 85
5. Сдвоявания в произволни графи . . . 91
6. Ребрени покрития в графи . . . 96
7. Върхови покрития в графи . . . 103
1.6. Ойлерови и Хамилтонови графи . . . 110
1. Ойлерови графи . . . 110
2. Хамилтонови графи . . . 118
1.7. Планарни и двойнствени графи. Оцветявания . . . 127
1. Планарни графи . . . 128
2. Формула на Ойлер . . . 130
3. Характеризация на Куратовски-Понтрягин за планарност . . . 136
4. Характеризация на Вагнер, Харари и Тат за планарност . . . 137
5. Характеризация на Маклейн . . . 138
6. Двойнствени графи . . . 138
7. Оцветявания . . . 140
8. Оцветяване на върховете . . . 141
9. Ребрено оцветяване. Хроматичен индекс . . . 149
1.8. Алгоритми за анализ на графи . . . 153
1. Машинно представяне на графи . . . 154
2. Търсене в дълбочина . . . 158
3. Търсене в ширина . . . 164
4. Принципи за търсене в дървото на решенията . . . 167
5. Използване на граници . . . 168
6. Търсене с връщане назад . . . 169

Литература **173**

2 Оптимизационни алгоритми в графи и мрежи **175**

2.1. Оптимизационни задачи. Линейно оптимиране . . . 175
1. Моделиране . . . 176
2. Типични класове оптимизационни задачи . . . 179
3. Задача на линейното оптимиране . . . 182
4. Канонична задача. Базис на опорен план . . . 187
5. Симплекс-метод . . . 192
6. Алгоритъм на симплекс-метода. Симплекс-таблици . . . 201
7. Симплекс-метод с изкуствен базис (М-задача) . . . 207
8. Дуална задача. Основна теорема на линейното оптимиране . 210
9. Теореми за двойнственост . . . 216
2.2. Алгоритми за построяване на покриващи дървета . . . 222
1. Алгоритъм за построяване на покриващо дърво . . . 224
2. Алгоритъм за построяване на максимален ориентиран лес (МОЛ) . . . 232
2.3. Алгоритми за търсене на пътища . . . 242
1.Най-кратък път (НКП) между два дадени върха s и t. Алгоритъм на Дийкстра . . . 243
2. Алгоритъм на Форд за НКП [16] . . . 247
3. Алгоритми за търсене на всички НКП. Алгоритми на Флойд и Данциг . . . 249
4. Алгоритъм за търсене на K-ти по дължина пътища между два върха s и t . . . 260
5. Задачи, свеждащи се до търсене на НКП — път с най-голяма пропускателна способност и най-надежден път . . . 263
2.4. Сложност на алгоритми и задачи . . . 266
1. Някои класове на сложност — $\mathcal{NP}$, $\mathcal{NPC}$, $\mathcal{NPI}$ и $\mathcal{NPH}$ . . . 273
2. Анализ на изчислителната сложност на някои алгоритми . . . 278
2.5. Потоци в мрежи . . . 282
1. Алгоритъм за търсене на увеличаваща (потока) верига . . . 285
2. Алгоритъм за търсене на максимален поток (Алгоритъм на Форд-Фалкерсон) . . . 289
3. Модификация на Едмондс и Карп . . . 299
4. Алгоритъм за търсене на максимален поток при няколко източника и стока . . . 304
5. Алгоритъм за търсене на поток с минимална стойност (Форд-Фалкерсон) . . . 305
6. Алгоритъм на дефекта . . . 317
7. Динамични потоци в мрежа . . . 324
2.6. Мрежово планиране и управление . . . 328
1. Метод за намиране на критичен път . . . 329
2. Топологическа сортировка на върховете в ориентирани ациклични графи . . . 330
3. Разчет на най-ранните срокове за настъпване на събития . . 334
4. Разчет на най-късните срокове . . . 334
5. Проекти с минимална стойност . . . 341
6. Обобщени мрежови графици . . . 345
2.7. Разполагане на обекти . . . 349
1. Търсене на центрове . . . 356
2. Търсене на медиани . . . 358
2.8. Оптимални сдвоявания и покрития . . . 359
1. Оптимални сдвоявания в биполярни графи . . . 361

.2. Унгарски алгоритъм 364
.3. Оптимални сдвоявания в произволни графи 373
.4. Оптимални покрития в графи 388
лгоритми за задачи от тип CPP 392
.1. CPP в неориентиран граф $G = (X, A)$. 393
.2. CPP в ориентиран граф $G = (X, E)$. 398
Алгоритми за задачи от тип TSP 403
.1. Задача от тип TSP. 403
.2. Методи за решаване на TSP. Долни граници 406
.3. **Branch-and-bound** алгоритми за **TSP** 413

тура **419**

л на термините **422**